DB2 Universal Database for z/OS

Version 8

Utility Guide and Reference

Information Management software

SC18-7427-07

DB2 Universal Database for z/OS

Version 8

Utility Guide and Reference

Information Management software

SC18-7427-07

Note
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 905.

Eighth Edition, Softcopy Only (June 2009)

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

This book contains usage information for the tasks of system administration, database administration, and operation. It presents detailed information about using utilities, specifying syntax (including keyword and parameter descriptions), and starting, stopping, and restarting utilities. This book also includes job control language (JCL) and control statements for each utility.

This information assumes that your DB2 subsystem is running in Version 8 new-function mode. New functions are available only in new-function mode, unless explicitly stated otherwise in the product documentation. A few general exceptions exist for utilities and for optimization. In most cases, new functions are not supported in compatibility mode unless noted. For utilities and optimization, new functions are available in compatibility mode unless noted. The new functions that are available in compatibility mode and enabling-new-function mode are almost identical, but some new functions are supported to provide easier migration. Exceptions to these general statements are noted in the information.

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 Chapter 2, "DB2 utilities packaging," on page 7 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 system administrators, database administrators, system operators, and application programmers of DB2 online and stand-alone utilities.

Recommendation: Familiarize yourself with DB2 UDB for z/OS prior to using this book.

Conventions and terminology used in this book

This section provides information about terms and conventions in this book.

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.

Naming conventions used in this book

This section describes naming conventions that are unique to commands and utilities.

When you use a parameter for an object that is created by SQL statements (for example, tables, table spaces, and indexes), identify the object by following the SQL syntactical naming conventions. See the description for naming conventions in *DB2 SQL Reference*.

In this book, characters are classified as *letters, digits,* or *special characters*.

- A *letter* is any one of the uppercase characters A through Z (including the three characters that are reserved in the United States as alphabetic extenders for national languages, #, @, and $.).
- A *digit* is any one of the characters 0 through 9.
- A *special character* is any character other than a letter or a digit.

See Chapter 2 of *DB2 SQL Reference* for an additional explanation of long identifiers, short identifiers, and location identifiers.

authorization-id
A short identifier of one to eight letters, digits, or the underscore that identifies a set of privileges. An authorization ID must begin with a letter.

connection-name
An identifier of one to eight characters that identifies an address space connection to DB2. A connection identifier is one of the following values:

- TSO (for DSN processes that run in TSO foreground).
- BATCH (for DSN processes that run in TSO batch).
- DB2CALL (for the call attachment facility (CAF)).
- The system identification name (for IMS and CICS processes).

See Part 4 (Volume 1) of *DB2 Administration Guide* for more information about managing DB2 connections.

correlation-id
An identifier of 1 to 12 characters that identifies a process within an address space connection. A correlation ID must begin with a letter.

A correlation ID can be one of the following values:

- The TSO logon identifier (for DSN processes that run in TSO foreground and for CAF processes).
- The job name (for DSN processes that run in TSO batch).
- The PST#.PSBNAME (for IMS processes).
- The entry identifier.thread_number.transaction_identifier (for CICS processes).

See Part 4 (Volume 1) of *DB2 Administration Guide* for more information about correlation IDs.

cursor-name
An identifier that designates a result set. Cursor names that are specified with the EXEC SQL and LOAD utilities cannot be longer than eight characters.

database-name
A short identifier that identifies a database. The identifier must start with a letter and must not include special characters.

data-set-name
An identifier of 1 to 44 characters that identifies a data set.

dbrm-member-name
An identifier of one to eight letters or digits that identifies a member of a partitioned data set.

A DBRM member name should not begin with DSN because of a potential conflict with DB2-provided DBRM member names. If you specify a DBRM member name that begins with DSN, DB2 issues a warning message.

dbrm-pds-name
An identifier of 1 to 44 characters that identifies a partitioned data set.

ddname
An identifier of one to eight characters that identifies the name of a DD statement.

hexadecimal-constant
A sequence of digits or any of the letters from A to F (uppercase or lowercase).

hexadecimal-string
An X followed by a sequence of characters that begins and ends with the string delimiter, an apostrophe. The characters between the string delimiters must be a hexadecimal number.

index-name
A qualified or unqualified name that identifies an index.

A qualified index name is a short identifier, followed by a period and a long identifier. The short identifier is the authorization ID that owns the index.

An unqualified index name is a long identifier with an implicit qualifier. The implicit qualifier is an authorization ID that is determined by the rules in Chapter 2 of *DB2 SQL Reference*.

If the index name contains a blank, the name must be enclosed in quotation marks when specified in a utility control statement.

location-name
A location identifier of 1 to 16 letters (but excluding the alphabetic extenders), digits, or the underscore that identifies an instance of a database management system. A location name must begin with a letter.

luname
An SQL short identifier of one to eight characters that identifies a logical unit name. A LU name must begin with a letter.

member-name
An identifier of one to eight letters (including the three alphabetic extenders) or digits that identifies a member of a partitioned data set.

A member name should not begin with DSN because of a potential conflict with DB2-provided member names. If you specify a member name that begins with DSN, DB2 issues a warning message.

qualifier-name
An SQL short identifier of one to eight letters, digits, or the underscore that identifies the implicit qualifier for unqualified table names, views, indexes, and aliases.

string
A sequence of characters that begins and ends with an apostrophe.

subsystem-name
An identifier that specifies the DB2 subsystem as it is known to the operating system.

table-name
A qualified or unqualified name that identifies a table.

A fully qualified table name is a three-part name. The first part is a location name that identifies the DBMS at which the table is stored. The second part is the authorization ID that identifies the owner of the table. The third part is a long identifier. A period must separate each of the parts.

A two-part table name is implicitly qualified by the location name of the current server. The first part is the authorization ID that identifies the owner of the table. The second part is an SQL long identifier. A period must separate the two parts.

A one-part or unqualified table name is a long identifier with two implicit qualifiers. The first implicit qualifier is the location name of the current server. The second implicit qualifier is an authorization ID, which is determined by the rules in Chapter 2 of *DB2 SQL Reference*.

If the table name contains a blank, the name must be enclosed in quotation marks when specified in a utility control statement.

table-space-name
A short identifier that identifies a table space of an identified database. The identifier must start with a letter and must not include special characters. If a database is not identified, a table space name specifies a table space of database DSNDB04.

utility-id
An identifier of 1 to 16 characters that uniquely identifies a utility process within DB2. A utility ID must begin with a letter. The remaining characters can be uppercase and lowercase letters, numbers 0 through 9, and the following characters: #, $, ., ¢, !, ¬, and @.

How to read the syntax diagrams

The following rules apply to the syntax diagrams that are used in this book:

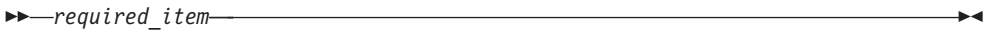
- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

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

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

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

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

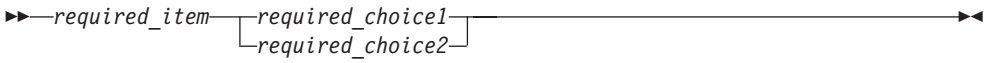


- Optional items appear below the main path.

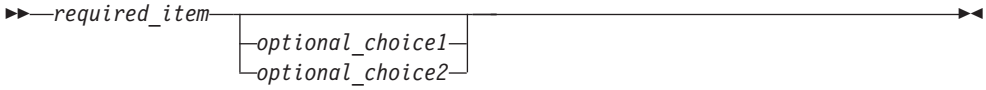
  If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

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

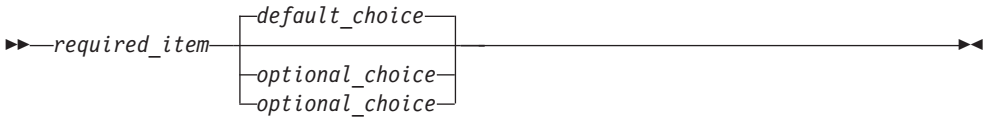
  If you *must* choose one of the items, one item of the stack appears on the main path.



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



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



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

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

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

- Keywords appear in uppercase (for example, `FROM`). They must be spelled exactly as shown. Variables appear in all lowercase letters (for example, *column-name*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

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 DB2 for z/OS - Technical Resources Web site at:

http://www.ibm.com/support/docview.wss?&uid=swg27011656

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.

Summary of changes to this book

DB2 UDB for z/OS, Version 8 includes several new online utilities. The new online utilities and their corresponding utility control statements are described in the following chapters:

DB2 UDB for z/OS, Version 8 changes to online utilities are included in the following chapters:

DB2 UDB for z/OS, Version 8 includes one new stand-alone utility, which is described in the following chapter:

DB2 UDB for z/OS, Version 8 changes to stand-alone utilities are included in the following chapters:

The following appendix is new for DB2 UDB for z/OS, Version 8:

The following appendixes have changed for DB2 UDB for z/OS, Version 8:

In addition, editorial changes have been made to the book.

All technical changes to the text are indicated by vertical bars (|) in the left margin.

Part 1. Introduction

Chapter 1. Basic information about the DB2 utilities

This chapter introduces the DB2 online and stand-alone utilities. This chapter also explains the authorization rules for coding utility control statements and the data sets that the utilities use.

The following topics provide additional information:

Types of DB2 utilities

The two types of DB2 utilities are online utilities and stand-alone utilities.

Description of online utilities

DB2 online utilities run as standard batch jobs or stored procedures, and they require DB2 to be running. They do not run under control of the terminal monitor program (TMP); they have their own attachment mechanisms. They invoke DB2 control facility services directly. See Chapter 3, "Invoking DB2 online utilities," on page 15 for information about the ways to run these utilities.

Description of stand-alone utilities

The stand-alone utilities run as batch jobs that are independent of DB2. The only way to run these utilities is to use JCL. See the chapters on the individual utilities in Chapter 33, "Invoking stand-alone utilities," on page 671 for information about the ways to run these utilities.

Privileges and authorization IDs

A command or a utility job can be issued by an individual user, by a program that runs in batch mode, or by an IMS or CICS transaction. The term *process* describes any of these initiators.

A process is represented to DB2 by a set of identifiers (IDs). What the process can do with DB2 is determined by privileges and privileges that can be held by its identifiers. The phrase *"privilege set of a process"* means the entire set of privileges and authorities that can be used by the process in a specific situation.

Three types of identifiers exist: primary authorization IDs, secondary authorization IDs, and SQL authorization IDs.

- Generally, the *primary authorization ID* identifies a specific process. For example, in the process that is initiated through the TSO attachment facility, the primary authorization ID is identical to the TSO logon ID. A trace record identifies the process by that ID.
- *Secondary authorization IDs*, which are optional, can hold additional privileges that are available to the process. A secondary authorization ID is often a SecureWay Security Server Resource Access Control Facility (RACF) group ID.

For example, a process can belong to a RACF group that holds the LOAD privilege on a particular database. Any member of the group can run the LOAD utility to load table spaces in the database.

DB2 commands that are entered from a z/OS console are not associated with any secondary authorization IDs.

- An *SQL authorization ID (SQL ID)* holds the privileges that are exercised when issuing certain dynamic SQL statements. Generally, this book does not discuss the SQL ID.

Within DB2, a process can be represented by a primary authorization ID and possibly one or more secondary IDs. For detailed instructions on how to associate a process with one or more IDs, and how to grant privileges to those IDs, see information about processing connections and sign-ons in Part 3 (Volume 1) of *DB2 Administration Guide*.

An administrator can grant or revoke a privilege or authority for an identifier by executing an SQL GRANT or a REVOKE statement. For the complete syntax of those statements, see Chapter 5 of *DB2 SQL Reference*.

If you use the access control authorization exit routine, that exit routine might control the authorization rules, rather than the exit routines that are documented for each utility.

Utilities that can be run on declared temporary objects

You can run the following two utilities on declared temporary objects:

- You can use the REPAIR DBD utility on declared temporary tables, which must be created in a database that is defined with the AS TEMP clause.
- # # You can use the STOSPACE utility on storage groups that have objects within temporary databases.

No other DB2 utilities can be used on a declared temporary table, its indexes, or its table spaces.

For detailed information about target object support, see the "Concurrency and compatibility" section in each utility chapter.

The effect of utilities on objects that have the DEFINE NO attribute

With DB2 Version 7 or above, you can run certain online utilities on table spaces or index spaces that were defined with the DEFINE NO attribute. When you specify this attribute, the table space or index space is defined, but DB2 does not allocate the associated data sets until a row is inserted or loaded into a table in that table space. For more information about the DEFINE NO attribute, see *DB2 Administration Guide*.

You can populate table spaces whose data sets are not yet defined by using the LOAD utility with either the RESUME keyword, the REPLACE keyword, or both. Using LOAD to populate these table spaces results in the following actions:

1. DB2 allocates the data sets.
2. DB2 updates the SPACE column in the catalog table to show that data sets exist.
3. DB2 loads the specified table space.

For a partitioned table space, all partitions are allocated even if the LOAD utility is loading only one partition. Avoid attempting to populate a partitioned table space with concurrent LOAD PART jobs until after one of the jobs has caused all the data sets to be created.

The following online utilities issue informational message DSNU185I when a table space or index space with the DEFINE NO attribute is encountered. The object is not processed.

- CHECK DATA
- CHECK INDEX
- COPY
- MERGECOPY
- MODIFY RECOVERY
- QUIESCE
- REBUILD INDEX
- RECOVER
- REORG INDEX
- REORG TABLESPACE
- REPAIR, but not REPAIR DBD
- RUNSTATS TABLESPACE INDEX(ALL) [1]
- RUNSTATS INDEX [1]
- UNLOAD

Notes:

1. RUNSTATS recognizes DEFINE NO objects and updates the catalog's access path statistics to reflect the empty objects.

Online utilities that encounter an undefined target object might issue informational message DSNU185I, but processing continues.

You cannot use stand-alone utilities on objects whose data sets have not been defined.

The effect of utilities on encrypted data

You can copy and recover encrypted data. You can also move encrypted data between systems. Data remains encrypted throughout these processes.

However, running any of the following utilities on encrypted data might produce unexpected results:

- CHECK DATA
- LOAD
- REBUILD INDEX
- REORG TABLESPACE
- REPAIR
- RUNSTATS
- UNLOAD
- DSN1PRNT

For more information about how each of these utilities processes encrypted data, see the individual chapters for each utility.

Chapter 2. DB2 utilities packaging

The following utilities are core utilities, which are included (at no extra charge) with Version 8 of DB2 UDB for z/OS:

- CATENFM
- CATMAINT
- DIAGNOSE
- LISTDEF
- OPTIONS
- QUIESCE
- REPAIR
- REPORT
- TEMPLATE
- All DSN stand-alone utilities

All other utilities are available as a separate product called the **DB2 Utilities Suite** (5655-K61, FMIDs JDB881K and JDB881M), which includes the following utilities:

- BACKUP SYSTEM
- CHECK DATA
- CHECK INDEX
- CHECK LOB
- COPY
- COPYTOCOPY
- EXEC SQL
- LOAD
- MERGECOPY
- MODIFY RECOVERY
- MODIFY STATISTICS
- REBUILD INDEX
- RECOVER
- REORG INDEX
- REORG TABLESPACE
- RESTORE SYSTEM
- RUNSTATS
- STOSPACE
- UNLOAD

All DB2 utilities operate on catalog, directory, and sample objects, without requiring any additional products.

The following topics provide additional information:

- "SMP/E jobs for DB2 utility products"
- "The operation of DB2 utilities in a mixed-release data sharing environment" on page 8

SMP/E jobs for DB2 utility products

To load the DB2 utility products, use System Modification Program Extended (SMP/E). SMP/E processes the installation tapes or cartridges and creates DB2 distribution target libraries.

DB2 provides several jobs that invoke SMP/E. These jobs are on the tape or cartridge that you received with the utility product. The job prologues in these jobs contain directions on how to tailor the job for your site. Follow these directions carefully to ensure that your DB2 UDB for z/OS SMP/E process works correctly. To copy the jobs from the tapes, submit the copy job that is listed in *DB2 Program Directory*.

The SMP/E RECEIVE job, DSNRECVS, loads the DB2 Diagnostic and Recovery Utilities program modules, macros, and procedures into temporary data sets (SMPTLIBs). The SMP/E RECEIVE job, DSNRECVK, loads the DB2 Operational Utilities program modules, macros, and procedures into temporary data sets (SMPTLIBs). If these jobs fail or abnormally terminate, correct the problem and rerun the jobs. Use job DSNRECV1, which is described in *DB2 Installation Guide*, as a guide to help you with the RECEIVE job.

The SMP/E APPLY job, DSNAPPLS, copies and link-edits the program modules, macros, and procedures for both the DB2 Diagnostic and Recovery Utilities and the DB2 Operational Utilities into the DB2 target libraries. Use job DSNAPPL1, which is described in *DB2 Installation Guide*, as a guide to help you with the APPLY job.

The SMP/E ACCEPT job, DSNACCPS, copies the program modules, macros, and procedures for both the DB2 Diagnostic and Recovery Utilities and the DB2 Operational Utilities into the DB2 distribution libraries. Use job DSNACEP1, which is described in *DB2 Installation Guide*, as a guide to help you with the ACCEPT job.

The operation of DB2 utilities in a mixed-release data sharing environment

The utilities batch module, DSNUTILB, is split into multiple parts: a release-independent module called DSNUTILB and multiple release-dependent modules, DSNUT810 and the utility-dependent load modules that are listed in Table 1. To operate in a mixed-release data sharing environment, you must have the release-dependent modules from both releases and all applicable utility-dependent modules available to the utility jobs that operate across the data sharing group. The procedure for sharing utility modules is explained in Chapter 4 of *DB2 Data Sharing: Planning and Administration*. Use the information in Table 1 and the procedures that are outlined in Chapter 4 of *DB2 Data Sharing: Planning and Administration* to implement a mixed-release data sharing environment.

With Version 7 and subsequent releases, each utility has separate load modules and aliases. Table 1. lists the alias name and load module name or names for each utility.

Table 1. Relationship between utility names, aliases, and load modules

Utility name	Alias name	Load module name
BACKUP SYSTEM and RESTORE SYSTEM	DSNU81AV	DSNU8RLV
CATMAINT and CATENFM [1]	DSNU81AA	DSNU8CLA
CHECK	DSNU81AB	DSNU8RLB
COPY	DSNU81AC	DSNU8OLC or DSNU8RLC
COPYTOCOPY	DSNU81AT	DSNU8RLT
DIAGNOSE	DSNU81AD	DSNU8CLD
EXEC SQL	DSNU81AU	DSNU8OLU

Table 1. Relationship between utility names, aliases, and load modules (continued)

Utility name	Alias name	Load module name
LISTDEF	DSNU81AE	DSNU8CLE
LOAD	DSNU81AF	DSNU8OLF
MERGECOPY	DSNU81AG	DSNU8RLG
MODIFY RECOVERY and MODIFY STATISTICS	DSNU81AH	DSNU8RLH
OPTIONS	DSNU81AI	DSNU8CLI
QUIESCE	DSNU81AJ	DSNU8CLJ
REBUILD INDEX	DSNU81AK	DSNU8OLK or DSNU8RLK
RECOVER	DSNU81AL	DSNU8OLL or DSNU8RLL
REORG INDEX and REORG TABLESPACE	DSNU81AM	DSNU8OLM
REPAIR	DSNU81AN	DSNU8CLN
REPORT	DSNU81AO	DSNU8CLO
RUNSTATS	DSNU81AP	DSNU8OLP
STOSPACE	DSNU81AQ	DSNU8OLQ
TEMPLATE	DSNU81AR	DSNU8CLR
UNLOAD	DSNU81AS	DSNU8OLS
Note: [1] The code for CATENFM is in the load module for CATMAINT.		

Part 2. DB2 online utilities

Chapter 3. Invoking DB2 online utilities

This chapter contains procedures and guidelines for creating utility control statements and describes five methods for invoking the DB2 utilities.

Creating utility control statements is the first step that is required to run an online utility.

After creating the utility statements, use one of the following methods for invoking the online utilities:

1. "Using the DB2 Utilities panel in DB2I" on page 24
2. "Using the DSNU CLIST command in TSO" on page 27
3. "Using the supplied JCL procedure (DSNUPROC)" on page 34
4. "Creating the JCL data set yourself by using the EXEC statement" on page 37
5. "Invoking utilities as a stored procedure (DSNUTILS)" on page 799 or "DSNUTILU stored procedure" on page 809

Requirement: In the JCL for all utility jobs, specify a load library that is at a maintenance level that is compatible with the DB2 system. Otherwise, errors can occur.

For the least involvement with JCL, use either the first or second method, and then edit the generated JCL to alter or add necessary fields on the JOB or ROUTE cards before submitting the job. Both of these methods require TSO, and the first method also requires access to the DB2 Utilities Panel in DB2 Interactive (DB2I).

If you want to work with JCL or create your own JCL, choose the third or fourth method.

To invoke online utilities from a DB2 application program, use the fifth method. For more information about these stored procedures and other stored procedures that are supplied by DB2, see Appendix B, "DB2-supplied stored procedures," on page 797.

The following topics provide additional information:

- "Creating utility control statements"
- "Data sets that online utilities use" on page 19
- "Extended addressing support by DB2 utilities" on page 23
- "Required authorizations for invoking online utilities on tables that have multilevel security with row-level granularity" on page 23
- "Using the DB2 Utilities panel in DB2I" on page 24
- "Using the DSNU CLIST command in TSO" on page 27
- "Using the supplied JCL procedure (DSNUPROC)" on page 34
- "Creating the JCL data set yourself by using the EXEC statement" on page 37

Creating utility control statements

Utility control statements define the function that the utility job performs.

Create the utility control statements with the ISPF/PDF edit function. Use the rules that are listed in "Control statement coding rules" on page 16.

After the utility control statements are created, save them in a sequential or partitioned data set.

Control statement coding rules

DB2 typically reads utility control statements from the SYSIN data set. DB2 can read LISTDEF control statements from the SYSLISTD data set and TEMPLATE control statements from the SYSTEMPL data set. The statements in these data sets must obey the following rules:

- If the records are 80-character fixed-length records, DB2 ignores columns 73 through 80.
- The records are concatenated before they are parsed; therefore, a statement or any of its syntactical constructs can span more than one record. No continuation character is necessary.
- All control statements in a given data set must be written entirely in a single character set. Two character sets are supported, EBCDIC (code page 500) or Unicode UTF-8 (code page 1208). DB2 automatically detects and processes Unicode UTF-8 control statements if the first character of the data set is:
  - A Unicode UTF-8 blank (x'20')
  - A Unicode UTF-8 dash (x'2D')
  - A Unicode UTF-8 upper case A through Z (x'41' through x'5A')

  In all other cases, the control statement data set is processed as EBCDIC. An informational message is issued to identify the character set that is being processed. If UTF-8 is used, quoted strings may not print correctly in utility error messages because all messages are printed in EBCDIC while the quoted strings remain in UTF-8.
- The control statements must begin with one of the following online utility or control statement names, and at least one blank must follow the name:
  - BACKUP SYSTEM
  - CATENFM
  - CATMAINT
  - CHECK DATA
  - CHECK INDEX
  - CHECK LOB
  - COPY
  - COPYTOCOPY
  - DIAGNOSE
  - EXEC SQL
  - LISTDEF
  - LOAD
  - MERGECOPY
  - MODIFY RECOVERY
  - MODIFY STATISTICS
  - OPTIONS
  - QUIESCE
  - REBUILD INDEX
  - RECOVER
  - REORG INDEX
  - REORG TABLESPACE
  - REPAIR
  - REPORT
  - RESTORE SYSTEM
  - RUNSTATS
  - STOSPACE

  - TEMPLATE
  - UNLOAD
- Other syntactical constructs in the utility control statement describe options; you can separate these constructs with an arbitrary number of blanks.
- The SYSIN stream can contain multiple utility control statements.

The options that you can specify after the online utility name depend on which online utility you use. To specify a utility option, specify the option keyword, followed by its associated parameter or parameters, if any. The parameter value can be a keyword. You need to enclose the values of some parameters in parentheses. The syntax diagrams for utility control statements that are included in this book show parentheses where they are required.

You can specify more than one utility control statement in the SYSIN stream. However, if any of the control statements returns a return code of 8 or greater, the subsequent statements in the job step are not executed.

When specifying in a utility control statement multiple numeric values that are meant to be delimited, you must delimit these values with a comma (″,″), regardless of the definition of DECIMAL in DSNHDECP. Likewise, when specifying a decimal number in a utility control statement, you must use a period (″.″), regardless of the definition of DECIMAL in DSNHDECP.

You can enter comments within the SYSIN stream. Comments must begin with two hyphens (**--**) and are subject to the following rules:

- You must use two hyphens on the same line with no space between them.
- You can start comments wherever a space is valid, except within a delimiter token.
- The end of a line terminates a comment.

Two comments are shown in the following statement:

```
// SYSIN DD *
RUNSTATS TABLESPACE DSNDB06.SYSDBASE  -- COMMENT HERE
-- COMMENT HERE
/*
```

Unicode character strings

When control statements are submitted in UNICODE they are translated into EBCDIC before processing. Quoted strings, however, are not translated and are processed exactly as they are entered. Depending on how the character string is being used, this may or may not be the desired behavior. To enter an EBCDIC character string in a UNICODE control statement you must use hexadecimal notation. This is often required when working with DATE, TIME and TIMESTAMP literals since they contain special characters and must be enclosed in quotes. For example:

- A DATE literal may be entered as X'20050901' which represents 'yyyy-mm-dd' packed
- A TIME literal may be entered as X'123059' which represents 'hh:mm:ss' packed
- A TIMESTAMP literal may be entered as X'20050901123059123456' which represents 'yyyy-mm-dd-hh.mm.ss.mmmmmm' packed

Tips for using multi-byte character sets

Multi-byte character sets can be difficult to work with in fixed 80-byte SYSIN data sets. Long object names and long character literals might not fit on a single line.

Where possible, avoid having to break object names or character literals:

- Use a SYSIN with variable length records or sufficiently large record length.
- Use shorter object names. The longer the name, the more likely there will be continuation issues.
- If possible, process the object by space name (tablespace or indexspace) and avoid using long multi-byte table and index names in utility syntax.

If necessary, use a continuation technique:

- Shift the starting point of the string left or right within the input record such that a complete multi-byte character ends in column 72. Continue with the next character in column 1 of the next input record.
- Separate qualified object names into two parts following the dot ″.″ which separates the qualifiers. Separating long names into multiple parts makes it easier to follow the continuation rules This technique can not be used in the EXEC SQL utility, which must follow both utility and SQL syntax rules.
- Use the || concatenation operator to divide long identifiers into two or more parts that fit properly into each SYSIN record. Place the || concatenation operator between two delimited character strings or between two non-delimited character strings. Delimited character strings are enclosed in double quotation marks. The || concatenation operator must be preceded and followed by at least one blank space. An example of the || concatenation operator is shown in the following statement:

  ```
  Example: LOAD INTO TABLE
           CREA ||
           TOR.
           "TABL" ||
           "ENAME"
  ```

  In this example, the strings CREA and TOR are non-delimited, and the strings TABL and ENAME are delimited by double quotation marks. The processed output of this example is equivalent to the following statement:

  ```
  LOAD INTO TABLE CREATOR."TABLENAME"
  ```

  The utility || operator is ignored in an EXEC SQL control statement by utility processing since the operator has an existing SQL meaning. The operators remain part of the SQL statement for subsequent processing by SQL.

Using the concatenation operator

Utility control statements support the || concatenation operator. The operator is allowed between two non-delimited character strings or two quoted character strings. The result is a character string consisting of the string following the operator concatenated after the string preceding the operator. The operation is shown in the following statement:

```
string1  ||  string2
```

Both *string1* and *string2* must be syntactically correct within each SYSIN input record. Quotes must be balanced within each string. If DBCS characters are used, shift-out and shift-in characters must be balanced within each string. Any one multi-byte character must be contained entirely within a single SYSIN record.

The || operator must be entered as a stand-alone token, with one or more blanks preceding and following it. It may be entered on the same input record as *″string1″*, alone or on an input record, or on the same input record with *″string2″*. This operator functions at the token level before any context is detected or semantic meaning is applied. An example utility statement is shown in the following statement:

```
COPY  INDEX
       "A" ||
       "B"
results in:
       COPY INDEX   "AB"
```

The utility || operator is ignored in an EXEC SQL control statement by utility processing since the operator has an existing SQL meaning. The operators remain part of the SQL statement for subsequent processing by SQL.

Descriptions of utility options

Where the syntax of each utility control statement is described, parameters are indented under the option keyword that they must follow. The following option is a typical example:

WORKDDN *ddname*
Specifies a temporary work file.

ddname is the data set name of the temporary file. The **default** is **SYSUT1**.

In the example, WORKDDN is an option keyword, and *ddname* is a variable parameter. As noted previously, you can enclose parameter values in parentheses, but parentheses are not always required. You can specify the temporary work file as either `WORKDDN SYSUT1` or `WORKDDN (SYSUT1)`.

Data sets that online utilities use

Every online utility job requires a SYSIN DD statement to describe an input data set; some utilities also require other data sets. Table 2 lists the name of each DD statement that might be needed, the online utilities that use it, and the purpose of the corresponding data sets. If a DD name other than one specified in the table is allowed, you specify it as a parameter in a utility option. Table 2 also lists the option keywords that you can use. DCB attributes that you specify on the DD statement are referred to as *user-specified values*.

Table 2. Data sets that online utilities use

DD name	Used by	Purpose	Option keyword
DATAWK*nn*	REORG	A work data set for sorting data, where *nn* is a two-digit number. You can use several data sets. To estimate the size of the data set that is needed, see "Data sets that REORG TABLESPACE uses" on page 453.	All REORGs except for REORGs of catalog and directory objects with links
ddname	COPY[1]	A single data set that DB2 uses when you specify the FILTERDDN option in the utility control statement; contains a list of VSAM data set names that are used during COPY jobs that use the CONCURRENT and FILTERDDN options.	FILTERDDN
DSSPRINT	COPY	An output data set for messages; required when CONCURRENT copy is used and the SYSPRINT DD card points to a data set.	CONCURRENT

Table 2. Data sets that online utilities use (continued)

DD name	Used by	Purpose	Option keyword
RNPRIN01	RUNSTATS	A data set that contains messages from DFSORT (usually, SYSOUT or DUMMY). This data set is used when distribution statistics are collected for column groups.	COLGROUP
SORTOUT	CHECK DATA[2,3] LOAD[3,4,5]	A data set that holds sorted keys (sort output) and allows the SORT phase to be restarted.	WORKDDN
SORTWK*nn*[6]	CHECK DATA, CHECK INDEX, CHECK LOB, LOAD, REBUILD INDEX, REORG	A work data set for sorting indexes, where *nn* is a two-digit number. You can use several data sets. To estimate the size of the data set that is needed, see 69 for CHECK DATA, 84 for CHECK INDEX, 237 for LOAD, 342 for REBUILD INDEX, or 456 for REORG.	None
ST01WK*nn*	LOAD, REBUILD INDEX, REORG INDEX, REORG TABLESPACE, RUNSTATS	A temporary data set for sort input and output when collecting statistics on at least one data-partitioned secondary index. Also used with STATWK01 for RUNSTATS with COLGROUP and FREQVAL option.	STATISTICS [7]
STATWK01	RUNSTATS	A temporary data set for sort input and output when collecting distribution statistics for column groups.	COLGROUP
STPRIN01	LOAD, REBUILD INDEX, REORG INDEX, REORG TABLESPACE	A data set that contains messages from DFSORT (usually, SYSOUT or DUMMY). This data set is used when statistics are collected on at least one data-partitioned secondary index.	STATISTICS
SW*mm*WK*nn*[6]	LOAD, REBUILD INDEX, REORG	An optional work data sets for sorting index keys by using the SORTKEYS keyword, where *mm* and *nn* are two-digit numbers. You can use several data sets. To estimate the size of the data set that is needed, see "Estimating the sort work file size" on page 260 for LOAD, "Estimating the sort work file size" on page 347 for REBUILD INDEX, or "Estimating the sort work file size" on page 474 for REORG.	None
SYSCOPY	COPY MERGECOPY LOAD[8] REORG[8]	An output data set for copies.	COPYDDN, RECOVERYDDN

Table 2. Data sets that online utilities use (continued)

DD name	Used by	Purpose	Option keyword
SYSDISC	LOAD, REORG DISCARD, optional for REORG	A data set that contains discarded records (optional).	DISCARDDN
SYSERR	CHECK DATA[2] LOAD	A data set that contains information about errors that are encountered during processing.	ERRDDN
SYSIN	All utilities	An input data set for utility statements.	None
SYSMAP	LOAD[5]	A data set that contains information about which input records violated a constraint.	MAPDDN
SYSPRINT	All utilities	A data set for messages and printed output (usually SYSOUT).	None
SYSPUNCH	REORG, UNLOAD	A data set that contains a LOAD statement that is generated by REORG or UNLOAD. For REORG, the LOAD statement loads records that REORG DISCARD or REORG UNLOAD EXTERNAL wrote to the DISCARD or UNLOAD data sets.	PUNCHDDN
SYSREC	LOAD[2] REORG[9] UNLOAD[2]	A data set that contains the input data set of LOAD, unloaded records for REORG, or UNLOAD.	INDDN, UNLDDN
SYSUT1	CHECK DATA[3] CHECK INDEX[2] LOAD[3,4,5] MERGECOPY	A temporary work data set that holds sorted keys for input to the SORT phase; for MERGECOPY, this data set holds intermediate merged output.	WORKDDN
UTPRIN*mm*	LOAD, REBUILD INDEX, REORG	Optional print message data sets, which are used when the SORTKEYS keyword is specified, where *mm* is a two-digit number.	None
UTPRINT	CHECK DATA, CHECK INDEX, CHECK LOB, LOAD, REORG, REBUILD INDEX	A data set that contains messages from DFSORT (usually, SYSOUT or DUMMY).	None

Table 2. Data sets that online utilities use (continued)

DD name	Used by	Purpose	Option keyword

Notes:

[1] If you specify FILTERDDN, you must supply a name; no default DD name exits.

[2] Required.

[3] Data sets cannot be shared between SORTOUT and SYSUT1. Sharing these data sets can cause unpredictable results.

[4] Required for tables with indexes.

[5] When referential constraints exist and ENFORCE(CONSTRAINTS) is specified.

[6] If tape is specified, the minimum key length of all indexes that are involved in the sort phase must be at least 6 bytes. This length, when added to the internally assigned 12-byte header, must be at least 18 bytes, as required by DFSORT.

[7] STATISTICS keyword does not apply to the RUNSTATS utility.

[8] Required for LOAD with COPYDDN or RECOVERYDDN and for REORG with COPYDDN, RECOVERYDDN, SHRLEVEL REFERENCE, or SHRLEVEL CHANGE.

[9] Required unless you specify NOSYSREC or SHRLEVEL CHANGE.

For input data sets, the online utilities use the logical record length (LRECL), the record format (RECFM) and the block size (BLKSIZE) with which the data set was created. Variable-spanned (VS) or variable-blocked-spanned (VBS) record formats are not allowed for utility input data sets. The only exception is for the LOAD utility, which accepts unloaded data in VBS format.

For output data sets, the online utilities determine both the logical record length and the record format. Any specified values for LRECL or RECFM are ignored. If you supply block size, that size is used; otherwise, the utility chooses a block size that is appropriate for the storage device. DB2 does not support the large block interface (LBI). Partitioned data sets (PDS) are not allowed for output data sets.

For both input and output data sets, the online utilities use the value that you supply for the number of buffers (BUFNO), with a maximum of 99 buffers. The default number of buffers is 20. The utilities set the number of channel programs equal to the number of buffers. The parameters that specify the buffer size (BUFSIZE) and the number of channel programs (NCP) are ignored. If you omit any DCB parameters, the utilities choose default values.

Increasing the number of buffers (BUFNO) can result in an increase in real storage utilization and page fixing below the 16-MB line.

Restriction: DB2 does not support the undefined record format (RECFM=U) for any data set.

Data set concatenation

DB2 utilities let you concatenate unlike input data sets. Therefore, the data sets in a concatenation list can have different block sizes, logical record lengths, and record formats. If you want to concatenate variable and fixed-blocked data sets, the logical record length must be 8 bytes smaller than the block size.

You cannot concatenate output data sets.

Controlling data set disposition

Most data sets need to exist only during utility execution (for example, during reorganization). However, you must keep several data sets in certain circumstances:

- Retain the image copy data sets until you no longer need them for recovery.
- Retain the unload data sets if you specify UNLOAD PAUSE, UNLOAD ONLY, UNLOAD EXTERNAL, or DISCARD for the REORG utility.
- Retain the SYSPUNCH data set if you specify UNLOAD EXTERNAL or DISCARD for the REORG utility until you no longer need the contents for subsequent loads.
- Retain the discard data set until you no longer need the contents for subsequent loads.

Because you might need to restart a utility, take the following precautions when defining the disposition of data sets:

- Use DISP=(NEW,CATLG,CATLG) or DISP=(MOD,CATLG) for data sets that you want to retain.
- Use DISP=(MOD,DELETE,CATLG) for data sets that you want to discard after utility execution.
- Use DISP=(NEW,DELETE) for DFSORT SORTWK*nn* data sets, or refer to *DFSORT Application Programming: Guide* for alternatives.
- Do not use temporary data set names.

See Table 150 on page 800 and Table 151 on page 801 for information about the default data dispositions that are specified for dynamically allocated data sets.

Preventing unauthorized access to data sets

To prevent unauthorized access to data sets (for example, image copies), you can protect the data sets with the Resource Access Control Facility (RACF) licensed program. To use a utility with a data set that is protected by RACF, you must be authorized to access the data set.

Extended addressing support by DB2 utilities

DB2 utilities support extended address volumes (EAV) for VSAM data sets. The maximum supported volume size is 256K cylinders.

Required authorizations for invoking online utilities on tables that have multilevel security with row-level granularity

If you use RACF access control with multilevel security, you need additional authorizations to run certain LOAD, UNLOAD, and REORG TABLESPACE jobs on tables that have multilevel security with row-level granularity. These authorizations are explained in the authorization section of each utility chapter. In z/OS V1R3 and z/OS V1R4, you cannot run these utilities on tables that have multilevel security with row-level granularity.

All other utilities ignore the row-level granularity. They check only for authorization to operate on the table space; they do not check row-level authorization. For more information about multilevel security, see Part 3 of *DB2 Administration Guide*.

Using the DB2 Utilities panel in DB2I

If you do not have much JCL knowledge, using the DB2 Utilities panel is probably the best way to execute the DB2 online utilities.

Restriction: You cannot use the DB2 Utilities panel in DB2I to submit a BACKUP SYSTEM job, a COPYTOCOPY job, a RESTORE SYSTEM job, or a COPY job for a list of objects (with or without the CONCURRENT keyword).

If your site does not have default JOB and ROUTE statements, you must edit the JCL to define them. If you edit the utility job before submitting it, you must use the ISPF editor and submit your job directly from the editor. Use the following procedure:

1. Create the utility control statement for the online utility that you intend to execute, and save it in a sequential or partitioned data set.

   For example, the following utility control statement specifies that the COPY utility is to make an incremental image copy of table space DSN8D81A.DSN8S81D with a SHRLEVEL value of CHANGE:

   ```
   COPY TABLESPACE DSN8D81A.DSN8S81D
     FULL NO
     SHRLEVEL CHANGE
   ```

   For the rest of this example, suppose that you save the statement in the default data set, UTIL.
2. From the ISPF Primary Option menu, select the DB2I menu.
3. On the DB2I Utilities panel, select the UTILITIES option. Items that you must specify are highlighted on the DB2 Utilities panel, as shown in Figure 1.

```
DSNEUP01                        DB2 UTILITIES
 ===>

 Select from the following:

  1 FUNCTION ===> EDITJCL   (SUBMIT job, EDITJCL, DISPLAY, TERMINATE)
  2 JOB ID   ===> TEMP      (A unique job identifier string)
  3 UTILITY  ===> COPY      (CHECK DATA, CHECK INDEX, CHECK LOB,

                             COPY, DIAGNOSE, LOAD, MERGE, MODIFY,
                             QUIESCE, REBUILD, RECOVER, REORG INDEX,
                             REORG LOB, REORG TABLESPACE, REPORT,
                             REPAIR, RUNSTATS, STOSPACE, UNLOAD.)

  4 STATEMENT DATA SET    ===> UTIL

 Specify restart or preview option, otherwise specify NO.

  5 RESTART  ===> NO         (NO, CURRENT, PHASE or PREVIEW)

  6 LISTDEF? (YES|NO) ===>          TEMPLATE? (YES|NO) ===>

  *  The data set names panel will be displayed when required by a utility.


 PRESS:  ENTER to process     END to exit    HELP for more information
```

Figure 1. DB2 Utilities panel

4. Fill in field 1 with the function that you want to execute. In this example, you want to submit the utility job, but you want to edit the JCL first, so specify EDITJCL. After you edit the JCL, you do not need to return to this panel to submit the job. Instead, type SUBMIT on the editor command line.

5. Ensure that Field 2 is a unique identifier for your utility job. The default value is TEMP. In this example, that value is satisfactory; leave it as is.
6. Fill in field 3 with the utility that you want to run. To indicate REORG TABLESPACE of a LOB table space, specify REORG LOB.

   In this example, specify COPY.
7. Fill in field 4 if you want to use an input data set other than the default data set. Unless you enclose the data set name between apostrophes, TSO adds your user identifier as a prefix. In this example, specify UTIL, which is the default data set.
8. Change field 5 if this job restarts a stopped utility or if you want to execute a utility in PREVIEW mode. In this example, leave the default value, NO.
9. Specify in field 6 whether you are using LISTDEF statements or TEMPLATE statements in this utility. If you specify YES for LISTDEF or TEMPLATE, DB2 displays the Control Statement Data Set Names panel, but the field entries are optional.
10. Press Enter.

If you specify COPY, LOAD, MERGECOPY, REORG TABLESPACE, or UNLOAD as the utility in field 3, you must complete the fields on the Data Set Names panel, as shown in Figure 2. In this example, COPY was specified.

If LISTDEF YES or TEMPLATE YES is specified, the Control Set Data Set Names panel is displayed, as shown in Figure 3 on page 26.

```
DSNEUP02                        DATA SET NAMES
 ===>

   Enter data set name for LOAD or REORG TABLESPACE:
1 RECDSN   ==>

   Enter data set name for
LOAD, REORG TABLESPACE or UNLOAD:
2 DISCDSN  ==>

   Enter output data sets for local/current site for COPY, MERGECOPY,
     LOAD, or REORG:
3 COPYDSN  ==> ABC
4 COPYDSN2 ==>

   Enter output data sets for recovery site for COPY, LOAD, or REORG:
5 RCPYDSN1 ==> ABC1
6 RCPYDSN2 ==>
   Enter output data sets for REORG or UNLOAD:
7 PUNCHDSN ==>
 PRESS:  ENTER to process     END to exit    HELP for more information
```

Figure 2. Data Set Names panel

If the Data Set Names panel is displayed, complete the following steps. If you do not specify COPY, LOAD, MERGECOPY, REORG TABLESPACE, or UNLOAD in field 3 of the DB2 Utilities panel, the Data Set Names panel is not displayed; skip this procedure and continue with Figure 3 on page 26.

1. Fill in field 1 if you are running LOAD, REORG, or UNLOAD. For LOAD, you must specify the data set name that contains the records that are to be loaded. For REORG or UNLOAD, you must specify the unload data set. In this example, you do not need to fill in field 1, because you are running COPY.
2. Fill in field 2 if you are running LOAD or REORG with discard processing, in which case you must specify a discard data set. In this example, you do not need to fill in field 2, because you are running COPY.

3. Fill in field 3 with the primary output data set name for the local site if you are running COPY, LOAD, or REORG, or with the current site if you are running MERGECOPY. The DD name that the panel generates for this field is SYSCOPY. This is an optional field for LOAD and for REORG with SHRLEVEL NONE; this field is required for COPY, for MERGECOPY, and for REORG with SHRLEVEL REFERENCE or CHANGE. In this example, the primary output data set name for the local site is ABC.
4. Fill in field 4 with the backup output data set name for the local site if you are running COPY, LOAD, or REORG, or the current site if you are running MERGECOPY. The DD name that the panel generates for this field is SYSCOPY2. This is an optional field. In this example, you do not need to fill in field 4.
5. Fill in field 5 with the primary output data set for the recovery site if you are running COPY, LOAD, or REORG. The DD name that the panel generates for this field is SYSRCOPY1. This is an optional field. In this example, the primary output data set name for the recovery site is ABC1.
6. Fill in field 6 with the backup output data set for the recovery site if you are running COPY, LOAD, or REORG. The DD name that the panel generates for this field is SYSRCOPY2. This field is optional. In this example, you do not need to fill in field 6.
7. Fill in field 7 with the output data set for the generated LOAD utility control statements if you are running REORG UNLOAD EXTERNAL, REORG DISCARD, or UNLOAD. The DD name that the panel generates for this field is SYSPUNCH. In this example, you do not need to fill in field 7.
8. Press Enter.

The Control Statement Data Set Names panel, which is shown in Figure 3, is displayed if either LISTDEF YES or TEMPLATE YES is specified on the DB2 Utilities panel.

```
DSNEUP03         CONTROL STATEMENT DATA SET NAMES            SSID:
===>

Enter the data set name for the LISTDEF data set (SYSLISTD DD):
 1 LISTDEF DSN  ===>
    OPTIONAL or IGNORED

Enter the data set name for the TEMPLATE data set (SYSTEMPL DD):
 2 TEMPLATE DSN ===>
    OPTIONAL or IGNORED


PRESS:  ENTER to process     END to exit   HELP for more information
```

Figure 3. Control Statement Data Set Names panel

1. Fill in field 1 to specify the data set that contains a LISTDEF control statement. The default is the SYSIN data set. This field is ignored if you specified NO in the LISTDEF? field in the DB2 Utilities panel.

   For information about using a LISTDEF control statement, see Chapter 15, “LISTDEF,” on page 173.
2. Fill in field 2 to specify the data set that contains a TEMPLATE. The default is the SYSIN data set. This field is ignored if you specified NO in the TEMPLATE? field in the DB2 Utilities panel.

   For information about using TEMPLATE, see Chapter 31, “TEMPLATE,” on page 593.

Using the DSNU CLIST command in TSO

You can initiate a DB2 online utility by invoking the DSNU CLIST command under TSO. The CLIST command generates the JCL data set that is required to execute the DSNUPROC procedure and to execute online utilities as batch jobs. When you use the CLIST command, you do not need to be concerned with details of the JCL data set.

Restriction: You cannot use the DSNU CLIST command to submit a COPY job for a list of objects (with or without the CONCURRENT keyword).

The CLIST command creates a job that performs only one utility operation. However, you can invoke the CLIST command for each utility operation that you need, and then edit and merge the outputs into one job or step.

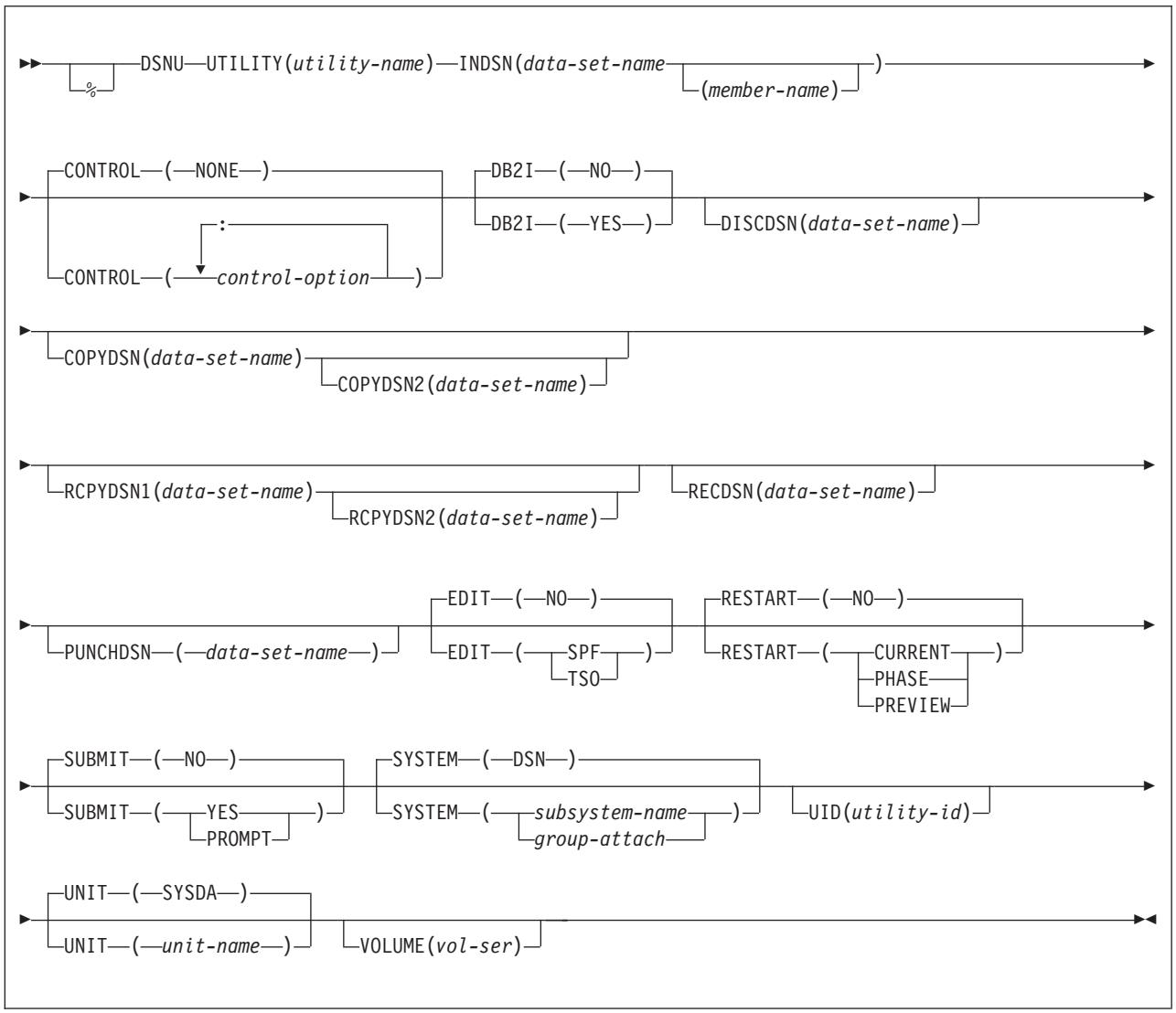
To use the DSNU CLIST command, follow these steps:

1. Create a file containing the required utility statements and control statements. DB2 uses the file to create the SYSIN data set in the generated job stream. Do not include double-byte character set (DBCS) data in this file.
2. Ensure that the DB2 CLIST library is allocated to the DD name SYSPROC.
3. Execute the command procedure by using the syntax in "DSNU CLIST command syntax" on page 28.
4. Edit the generated JCL data set to alter or add DD statements as needed.

   This last step is optional. "Editing the generated JCL data set" on page 33 explains how to edit the JCL data set.

You can execute the DSNU CLIST command from the TSO command processor or from the DB2I Utilities panel.

DSNU CLIST command syntax



DSNU CLIST option descriptions

The parentheses that are shown in the following descriptions are required. If you make syntax errors or omit parameter values, TSO prompts you for the correct parameter spelling and omitted values.

% Identifies DSNU as a member of a command procedure library. Specifying this parameter is optional; however, it does improve performance.

UTILITY (*utility-name***)**
Specifies the utility that you want to execute. Select the name from the following list:

- CHECK DATA
- CHECK INDEX
- CHECK LOB
- COPY
- DIAGNOSE
- LOAD
- MERGE

- MODIFY
- QUIESCE
- REBUILD
- RECOVER
- REORG INDEX
- REORG LOB
- REORG TABLESPACE
- REPAIR
- REPORT
- RUNSTATS
- STOSPACE
- UNLOAD

DB2 places the JCL in a data set that is named DSNU*xxx*.CNTL, where DSNU*xxx* is a control file name. The control file contains the statements that are necessary to invoke the DSNUPROC procedure which, in turn, executes the utility. If you execute another job with the same utility name, the first job is deleted. See "UID" on page 31 for a list of the online utilities and the control file name that is associated with each utility.

INDSN(*data-set-name***(***member-name***))**
Specifies the data set that contains the utility statements and control statements. Do not specify a data set that contains double-byte character set data.

(*data-set-name***)**
Specifies the name of the data set. If you do not specify a data set name, the default command procedure prompts you for the data set name.

(*member-name***)**
Specifies the member name. You must specify the member name if the data set is partitioned.

CONTROL(*control-option***: ...)**
Specifies whether to trace the CLIST command execution.

NONE
Omits tracing. The **default** is **NONE**.

control-option
Lists one or more of the following options. Separate items in the list by colons (:). To abbreviate, specify only the first letter of the option.

LIST Displays TSO commands after symbolic substitution and before command execution.

CONLIST
Displays CLIST commands after symbolic substitution and before command execution.

SYMLIST
Displays all executable statements (TSO commands and CLIST statements) before the scan for symbolic substitution.

NONE
Generates a CONTROL statement with the options NOLIST, NOCONLIST, and NOSYMLIST.

DB2I
Indicates the environment from which the DSNU CLIST command is called.

(NO) Indicates that DSNU CLIST command is not being called from the DB2I environment. The **default** is **NO**.

(YES) Indicates that DSNU CLIST command is called from the DB2I environment. Only the utility panels should execute the CLIST command with DB2I(YES).

DISCDSN (*data-set-name***)**
The name of the cataloged data set that LOAD and REORG use for a discard data set. For LOAD, this data set holds records that are not loaded; for REORG, it holds records that are not reloaded.

COPYDSN(*data-set-name***)**
The name of the cataloged data set that DB2 utilities use as a target (output) data set. If you do not supply this information, the CLIST command prompts you for it. This keyword is optional for LOAD and for REORG with SHRLEVEL NONE; it is required for COPY, for MERGECOPY, and for REORG with SHRLEVEL REFERENCE or CHANGE.

COPYDSN2 (*data-set-name***)**
The name of the cataloged data set that DB2 utilities use as a target (output) data set for the backup copy. This keyword is optional for COPY, MERGECOPY, LOAD, and REORG.

RCPYDSN1 (*data-set-name***)**
The name of the cataloged data set that DB2 utilities use as a target (output) data set for the remote-site primary copy. This keyword is optional for COPY, LOAD, and REORG.

RCPYDSN2 (*data-set-name***)**
The name of the cataloged data set that DB2 utilities use as a target (output) data set for the remote-site backup copy. This keyword is optional for COPY, LOAD, and REORG.

RECDSN (*data-set-name***)**
The name of the cataloged data set that LOAD uses for input or that REORG TABLESPACE or UNLOAD use as the unload data set. If you do not supply this information, the CLIST command prompts you for it. This keyword is required for LOAD and REORG TABLESPACE only.

PUNCHDSN (*data-set-name***)**
The name of the cataloged data set that REORG or UNLOAD use to hold the generated LOAD utility control statements for UNLOAD EXTERNAL or DISCARD.

EDIT
Specifies whether to invoke an editor to edit the temporary file that the CLIST command generates.

(NO)
Does not invoke an editor. The **default** is **NO**.

(SPF)
Invokes the ISPF editor.

(TSO)
Invokes the TSO editor.

RESTART
Specifies whether this job restarts a current utility job, and, if so, at what point it is to be restarted.

(NO)
Indicates that the utility is a new job, not a restarted job. The utility identifier (UID) must be unique for each utility job step. The **default** is **NO**.

(CURRENT)
Restarts the utility at the most recent commit point.

(PHASE)
Restarts the utility at the beginning of the current stopped phase. You can determine the current stopped phase by issuing the DISPLAY UTILITY command.

(PREVIEW)
Restarts the utility in PREVIEW mode. While in PREVIEW mode, the utility checks for syntax errors in all utility control statements, but normal utility execution does not take place.

SUBMIT
Specifies whether to submit the generated JCL for processing.

(NO)
Does not submit the JCL data set for processing. The **default** is **NO**.

(YES)
Submits the JCL data set for background processing, using the TSO SUBMIT command.

(PROMPT)
Prompts you, after the data set is processed, to specify whether to submit the JCL data set for batch processing. You cannot use PROMPT when the CLIST command is executed in the TSO batch environment.

SYSTEM (*subsystem-name***)**
Specifies the DB2 subsystem or group attach name. The **default** is **DSN**.

UID (*utility-id***)**
Provides a unique identifier for this utility job within DB2. Do not reuse the utility ID of a stopped utility that has not yet been terminated, unless you want to restart that utility. If you do use the same utility ID to invoke a different utility, DB2 tries to restart the original stopped utility with the information that is stored in the SYSUTIL directory table.

The **default** is *tso-userid.control-file-name*, where *control-file-name* for each of the utilities is listed Table 3.

Table 3. Control-file name for each utility

Utility	*control-file-name*
CHECK INDEX	DSNUCHI
CHECK DATA	DSNUCHD
CHECK LOB	DSNUCHL
COPY	DSNUCOP
DIAGNOSE	DSNUDIA
LOAD	DSNULOA
MERGECOPY	DSNUMER
MODIFY	DSNUMOD
QUIESCE	DSNUQUI

Table 3. Control-file name for each utility (continued)

Utility	*control-file-name*
REBUILD INDEX	DSNUREB
RECOVER	DSNUREC
REORG INDEX	DSNURGI
REORG LOB	DSNURGL
REORG TABLESPACE	DSNURGT
REPAIR	DSNUREP
REPORT	DSNURPT
RUNSTATS	DSNURUN
STOSPACE	DSNUSTO
UNLOAD	DSNUUNL

UNIT (*unit-name***)**
Assigns a unit address, a generic device type, or a user-assigned group name for a device on which a new temporary or permanent data set resides. When the CLIST command generates the JCL, it places *unit-name* after the UNIT clause of the generated DD statement. The **default** is **SYSDA**.

VOLUME (*vol-ser***)**
Assigns the serial number of the volume on which a new temporary or permanent data set resides. When the CLIST command generates the JCL, it places *vol-ser* after the VOL=SER clause of the generated DD statement. If you omit VOLUME, the VOL=SER clause is omitted from the generated DD statement.

DSNU CLIST command output

DSNU builds a one-step job stream. The JCL data set consists of a JOB statement, an EXEC statement that executes the DB2 utility processor, and the required DD statements. This JOB statement also includes the SYSIN DD * job stream, as shown in Figure 4. You can edit any of these statements.

```
//DSNUCOP JOB your-job-statement-parameters
//          USER=userid,PASSWORD=password
//*ROUTE PRINT routing-information
//UTIL      EXEC  DSNUPROC,SYSTEM=DSN,UID=TEMP,UTPROC='
//SYSCOPY   DD    DSN=MYCOPIES.DSN8D81A.JAN1,DISP=(MOD,CATLG,CATLG),
//          UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN     DD    *
     COPY TABLESPACE DSN8D81A.DSN8S81D
          FULL NO
          SHRLEVEL CHANGE

/*
```

Figure 4. Control file DSNUCOP.CNTL. This is an example of the JCL data set before editing.

The following list describes the required JCL data set statements:

Statement	Description
JOB	The CLIST command uses any JOB statements that you saved when using DB2I. If no JOB statements exist, DB2 produces a

skeleton JOB statement that you can modify. The job name is DSNU, followed by the first three letters of the name of the utility that you are using.

EXEC The CLIST command builds the EXEC statement. The values that you specified for SYSTEM (DSN, by default), UID(TEMP), and RESTART (none) become the values of SYSTEM, UID, and UTPROC for the DSNUPROC.

The CLIST command builds the necessary JCL DD statements. Those statements vary depending on the utility that you execute. Data sets that might be required are listed under "Data sets that online utilities use" on page 19. The following DD statements are generated by the CLIST command:

SYSPRINT DD SYSOUT=A
Defines OUTPUT, SYSPRINT as SYSOUT=A. Utility messages are sent to the SYSPRINT data set. You can use the TSO command to control the disposition of the SYSPRINT data set. For example, you can send the data set to your terminal. For more information, see *z/OS TSO/E Command Reference.*

UTPRINT DD SYSOUT=A
Defines UTPRINT as SYSOUT=A. If any utility requires a sort, it executes DFSORT. Messages from that program are sent to UTPRINT.

SYSIN DD *
Defines SYSIN. To build the SYSIN DD * job stream, DSNU copies the data set that is named by the INDSN parameter. The INDSN data set does not change, and you can reuse it when the DSNU procedure has finished running.

Editing the generated JCL data set

You can edit the data set before you process it by using the EDIT parameter on the command procedure. Use the editor to add a JCL statement to the job stream, to change JCL parameters (such as *ddnames*), or to change utility control statements.

If you use a *ddname* that is not the default on a utility statement that you use, you must change the *ddname* in the JCL that is generated by the DSNU procedure. For example, in the REORG TABLESPACE utility, the default option for UNLDDN is SYSREC, and DSNU builds a SYSREC DD statement for REORG TABLESPACE. If you use a different value for UNLDDN, you must edit the JCL data set and change SYSREC to the *ddname* that you used.

When you finish editing the data set, you can either save changes to the data set (by issuing SAVE), or instruct the editor to ignore all changes.

The SUBMIT parameter specifies whether to submit the data set statement as a background job. The temporary data set that holds the JCL statement is reused. If you want to submit more than one job that executes the same utility, you must rename the JCL data sets and submit them separately.

Examples

Example 1: The following CLIST command statement generates a data set that is called *authorization-id*.DSNURGT.CNTL and that contains JCL statements that invoke the DSNUPROC procedure.

```
%DSNU UTILITY(REORG TABLESPACE) INDSN(MYREOR.DATA)
  RECDSN(MYREOR.WORK) RESTART(NO)
  EDIT(TSO) SUBMIT(YES)
```

The DSNUPROC procedure invokes the REORG TABLESPACE utility. The MYREOR.DATA data set is merged into the JCL data set as SYSIN input. MYREOR.WORK is a temporary data set that is required by REORG TABLESPACE. The TSO editor is invoked to allow editing of the JCL data set, *authorization-id*.DSNURGT.CNTL. The TSO editor then submits the JCL data set as a batch job. This JCL data set is not modified by this CLIST command statement until a new request is made to execute the REORG TABLESPACE utility.

Example 2: The following example shows how to invoke the CLIST command for the COPY utility.

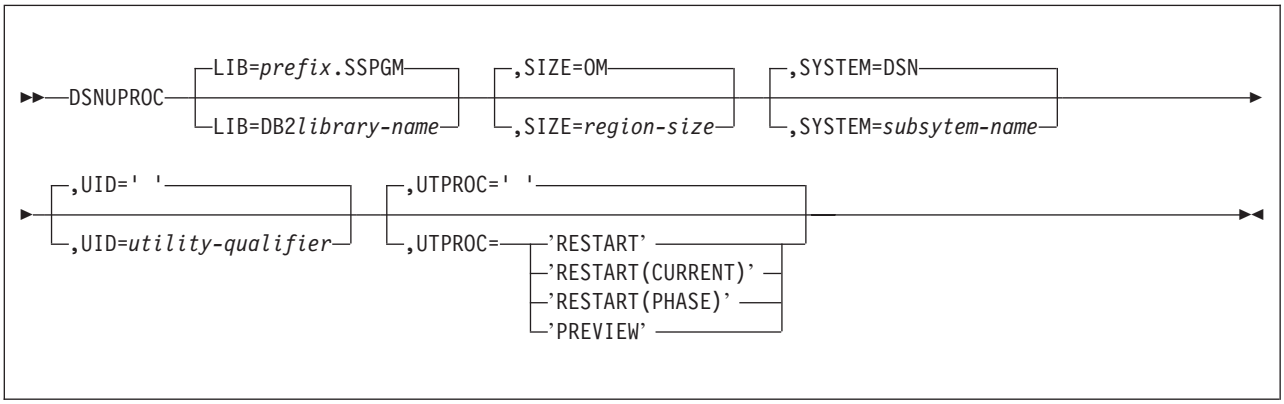
```
%DSNU
  UTILITY (COPY)
  INDSN ('MYCOPY(STATEMNT)')
  COPYDSN ('MYCOPIES.DSN8D81A.JAN1')
  EDIT (TSO)
  SUBMIT (YES)
  UID (TEMP)
  RESTART (NO)
```

Using the supplied JCL procedure (DSNUPROC)

Another method of invoking a DB2 online utility uses the supplied JCL procedure, DSNUPROC, which is shown in Figure 5 on page 36. This procedure uses the parameters that you supply to build an appropriate EXEC statement that executes an online utility.

To execute the DSNUPROC procedure, write and submit a JCL data set like the one that the DSNU CLIST command builds (An example is shown in Figure 4 on page 32.) In your JCL, the EXEC statement executes the DSNUPROC procedure.

DSNUPROC syntax



DSNUPROC option descriptions

The following list describes all the parameters. For example, in Figure 4 on page 32, you need to use only one parameter, UID=TEMP; for all others, you can use the default values.

LIB= Specifies the data set name of the DB2 subsystem library. The **default** is *prefix*.**SSPGM**.

SIZE=

Specifies the region size of the utility execution area; that is, the value represents the number of bytes of virtual storage that are allocated to this utility job. The **default** is **0M**.

SYSTEM=
Specifies the DB2 subsystem or group attach name. The **default** is **DSN**.

UID= Specifies the unique identifier for your utility job. The maximum name length is 16 characters. If the name contains special characters, enclose the entire name between apostrophes (for example, 'PETERS.JOB'). Do not reuse the utility ID of a stopped utility that has not yet been terminated. If you do use the same utility ID to invoke a different utility, DB2 tries to restart the original stopped utility with the information that is stored in the SYSUTIL directory table.

The **default** is an empty string.

UTPROC=
Controls restart processing. The **default** is an empty string. Use the default if you do not want to restart a stopped job.

To restart the utility, specify:

'RESTART'
To restart at the most recent commit point. This option has the same meaning as 'RESTART(CURRENT).'

'RESTART(CURRENT)'
To restart at the most recent commit point. This option has the same meaning as 'RESTART.'

'RESTART(PHASE)'
To restart at the beginning of the phase that executed most recently.

'PREVIEW'
To restart in preview mode. While in PREVIEW mode, the utility checks for syntax errors in all utility control statements, but normal utility execution does not take place.

The DSNUPROC procedure provides the SYSPRINT and UTPRINT DD statements for printed output. You must provide DD statements for SYSIN and other data sets that your job needs. See "Data sets that online utilities use" on page 19 for a description of data sets that you might need.

Sample DSNUPROC listing

Figure 5 on page 36 is the DSNUPROC procedure that was executed by the JCL example in Figure 4 on page 32.

```
//DSNUPROC PROC LIB='DSN!!0.SDSNLOAD',
//         SYSTEM=DSN,
//         SIZE=0K,UID='',UTPROC=''
//*****************************************************************
//* PROCEDURE-NAME:      DSNUPROC                                 *
//*                                                               *
//* DESCRIPTIVE-NAME:    UTILITY PROCEDURE                        *
//*                                                               *
//* FUNCTION:  THIS PROCEDURE INVOKES THE ADMF UTILITIES IN THE   *
//*             BATCH ENVIRONMENT                                 *
//*                                                               *
//* PROCEDURE-OWNER:     UTILITY COMPONENT                        *
//*                                                               *
//* COMPONENT-INVOKED:   ADMF UTILITIES (ENTRY POINT DSNUTILB).   *
//*                                                               *
//* ENVIRONMENT:         BATCH                                    *
//*                                                               *
//* INPUT:                                                        *
//*      PARAMETERS:                                              *
//*           LIB    = THE DATA SET NAME OF THE DB2  PROGRAM LIBRARY.  *
//*                    THE DEFAULT LIBRARY NAME IS PREFIX.SDSNLOAD,    *
//*                    WITH PREFIX SET DURING INSTALLATION.            *
//*           SIZE   = THE REGION SIZE OF THE UTILITIES EXECUTION AREA.*
//*                    THE DEFAULT REGION SIZE IS 2048K.               *
//*           SYSTEM = THE SUBSYSTEM NAME USED TO IDENTIFY THIS JOB    *
//*                    TO DB2.  THE DEFAULT IS "DSN".                  *
//*           UID    = THE IDENTIFIER WHICH WILL DEFINE THIS UTILITY   *
//*                    JOB TO DB2.  IF THE PARAMETER IS DEFAULTED OR   *
//*                    SET TO A NULL STRING, THE UTILITY FUNCTION WILL *
//*                    USE ITS DEFAULT, USERID.JOBNAME.  EACH UTILITY  *
//*                    WHICH HAS STARTED AND IS NOT YET TERMINATED     *
//*                    (MAY NOT BE RUNNING) MUST HAVE A UNIQUE UID.    *
//*           UTPROC = AN OPTIONAL INDICATOR USED TO DETERMINE WHETHER *
//*                    THE USER WISHES TO INITIALLY START THE REQUESTED*
//*                    UTILITY OR TO RESTART A PREVIOUS EXECUTION OF   *
//*                    THE UTILITY.  IF OMITTED, THE UTILITY WILL      *
//*                    BE INITIALLY STARTED.  OTHERWISE, THE UTILITY   *
//*                    WILL BE RESTARTED BY ENTERING THE FOLLOWING     *
//*                    VALUES:                                         *
//*                       RESTART(PHASE)  = RESTART THE UTILITY AT THE *
//*                               BEGINNING OF THE PHASE EXECUTED      *
//*                               LAST.                                *
//*                          RESTART = RESTART THE UTILITY AT THE LAST   *
//*                               OR CURRENT COMMIT POINT.             *
//*                                                                    *
//* OUTPUT: NONE.                                                      *
//*                                                                    *
//* EXTERNAL-REFERENCES: NONE.                                         *
//*                                                                    *
//* CHANGE-ACTIVITY:                                                   *
//*                                                                    *
//*********************************************************************
//DSNUPROC EXEC PGM=DSNUTILB,REGION=&SIZE,
//         PARM='&SYSTEM,&UID,&UTPROC'
//STEPLIB  DD   DSN=&LIB,DISP=SHR
//*********************************************************************
//*                                                                   *
//*  THE FOLLOWING DEFINE THE UTILITIES' PRINT DATA SETS              *
//*                                                                   *
//*********************************************************************
//*
//SYSPRINT DD   SYSOUT=*
//UTPRINT  DD   SYSOUT=*
//SYSUDUMP DD   SYSOUT=*
//*DSNUPROC PEND        REMOVE * FOR USE AS INSTREAM PROCEDURE
```

Figure 5. Sample listing of supplied JCL procedure DSNUPROC

Creating the JCL data set yourself by using the EXEC statement

DB2 online utilities execute as standard z/OS jobs. To execute the utility, you must supply the JOB statement that is required by your installation and the JOBLIB or STEPLIB DD statements that are required to access DB2. You must also include an EXEC statement and a set of DD statements. The EXEC statement and the DD statements that you might need are described in "Data sets that online utilities use" on page 19.

Recommendation: Use DSNUPROC to invoke a DB2 online utility, rather than creating the JCL yourself. For more information, see "Using the supplied JCL procedure (DSNUPROC)" on page 34.

The EXEC statement can be a procedure that contains the required JCL, or it can be of the following form:

```
//stepname  EXEC PGM=DSNUTILB,PARM='system,[uid],[utproc]'
```

The brackets, [ ], indicate optional parameters. The parameters have the following meanings:

DSNUTILB
Specifies the utility control program. The program must reside in an APF-authorized library.

system Specifies the DB2 subsystem.

uid The unique identifier for your utility job. Do not reuse the utility ID of a stopped utility that has not yet been terminated. If you do use the same utility ID to invoke a different utility, DB2 tries to restart the original stopped utility with the information that is stored in the SYSUTIL directory table.

utproc The value of the UTPROC parameter in the DSNUPROC procedure. Specify this option only when you want to restart the utility job. Specify:

'RESTART'
To restart at the most recent commit point. This option has the same meaning as 'RESTART(CURRENT).'

'RESTART(CURRENT)'
To restart the utility at the most recent commit point. This option has the same meaning as 'RESTART.'

'RESTART(PHASE)'
To restart at the beginning of the phase that executed most recently.

'RESTART(PREVIEW)'
To restart the utility in preview mode. While in PREVIEW mode, the utility checks for syntax errors in all utility control statements, but normal utility execution does not take place.

For the example in Figure 5 on page 36 you can use the following EXEC statement:

```
//stepname
  EXEC PGM=DSNUTILB,PARM='DSN,TEMP'
```

Chapter 4. Monitoring and controlling online utilities

This section contains procedures and guidelines for monitoring utilities, running utilities concurrently, terminating utilities, and restarting utilities.

The following topics provide additional information:

Monitoring utilities with the DISPLAY UTILITY command

The information under this heading, up to "Running utilities concurrently" on page 41 is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 905.

Use the DB2 DISPLAY UTILITY command to check the current status of online utilities. Figure 6 shows an example of the output that the DISPLAY UTILITY command generates. In the example output, DB2 returns a message that indicates the member name (**A**), utility identifier (**B**), utility name (**C**), utility phase (**D**), the number of pages or records that are processed by the utility[1] (**E**), the number of objects in the list (**F**), the last object that started (**G**), and the utility status (**H**). The output might also report additional information about an executing utility, such as log phase estimates or utility subtask activity.

```
DSNU100I - DSNUGDIS - USERID = SAMPID
           A   MEMBER = DB1G
           B   UTILID = RUNTS
               PROCESSING UTILITY STATEMENT 1
           C UTILITY = RUNSTATS
           D   PHASE = RUNSTATS   E  COUNT = 0
           F       NUMBER OF OBJECTS IN LIST = n
           G       LAST OBJECT STARTED = m
           H  STATUS = STOPPED
DSN9022I - DSNUGCC  '-DISPLAY UTILITY' NORMAL COMPLETION
```

Figure 6. DISPLAY UTILITY command sample output

Determining the status of a utility

To determine the status of an online utility, look at the status part (**H**) of the DISPLAY UTILITY output. An online utility can have one of these statuses:

Status	Description
Active	The utility has started execution.
Stopped	The utility has abnormally stopped executing before completion, but the table spaces and indexes that were accessed by the utility

1. In a data sharing environment, the number of records is current when the command is issued from the same member on which the utility is executing. When the command is issued from a different member, the count might lag substantially. When the command is issued from a different member, the count might lag substantially. For some utilities in some phases, the count number is not updated when the command is issued from a different member.

remain under utility control. To make the data available again, you must take one of the following actions:

- Correct the condition that stopped the utility, and restart the utility so that it runs to termination.
- Terminate the utility with the DB2 TERM UTILITY command. (This command is described in "Terminating an online utility with the TERM UTILITY command" on page 42.)

Terminated The utility has been requested to terminate by the DB2 TERM UTILITY command. If the utility has terminated, no message is issued.

Determining which utility phase is currently executing

DB2 online utility execution is divided into phases. Each utility starts with the UTILINIT phase, which performs initialization and set up. Each utility finishes with a UTILTERM phase, which cleans up after processing has completed. The other phases of online utility execution differ, depending on the utility. See the "Execution Phases" section in the descriptions of each utility. To determine which utility phase is currently executing, look at the output from the DISPLAY UTILITY command. The example output in Figure 6 on page 39 shows the current phase (**D**).

Determining why a utility failed to complete

If an online utility job completes normally, it issues return code 0. If it completes with warning messages, it issues return code 4. Return code 8 means that the job failed to complete. Return code 12 means that an authorization error occurred.

To determine why a utility failed to complete, consider the following problems that can cause a failure during execution of the utility:

- **Problem:** DB2 terminates the utility job step and any subsequent utility steps.

  Solution: Submit a new utility job to execute the terminated steps. Use the same utility identifier for the new job to ensure that no duplicate utility job is running.
- **Problem:** DB2 does not execute the particular utility function, but prior utility functions are executed.

  Solution: Submit a new utility step to execute the function.
- **Problem:** DB2 places the utility function in the stopped state.

  Solution: Restart the utility job step at either the last commit point or the beginning of the phase by using the same utility identifier. Alternatively, use a TERM UTILITY (uid) command to terminate the job step and resubmit it.
- **Problem:** DB2 terminates the utility and issues return code 8.

  Solution: One or more objects might be in a restrictive or advisory status. See Appendix C, "Advisory or restrictive states," on page 853 for more information on resetting the status of an object.

  Alternatively, a DEADLINE condition in online REORG might have terminated the reorganization.

For more information about the DEADLINE condition, see the description of this option in Chapter 24, "REORG INDEX," on page 389 or in Chapter 25, "REORG TABLESPACE," on page 417.

Running utilities concurrently

Some online utilities allow other utilities and SQL statements to run concurrently on the same target object. To determine if utilities can be run concurrently, look in the compatibility and concurrency section in each online utility chapter. Each concurrency and compatibility section includes the following information:

- For each target object on which the utility acts, the section outlines the claim classes that the utility must claim or drain. The section also outlines the restrictive state that the utility sets on the target object.
- For other online utilities, the section summarizes the compatibility of the utility with the same target object. If two actions are compatible on a target object, they can run simultaneously on that object in separate applications. If compatibility depends on particular options of a utility, that dependency is also shown.

If the utility supports parallelism, it can use additional threads to support the parallel subtasking. Consider increasing the values of subsystem parameters that control threads, such as MAX BATCH CONNECT and MAX USERS. These parameters are on installation panel DSNTIPE and are described in *DB2 Installation Guide*.

See Part 5 (Volume 2) of *DB2 Administration Guide* for a description of the claim classes and the use of claims and drains by online utilities.

Running online utilities in a data sharing environment

This section discusses considerations for running online utilities in a data sharing environment.

Submitting online utility jobs: When you submit a utility job, you must specify the name of the DB2 subsystem to which the utility is to attach or the group attach name. If you do not use the group attach name, the utility job must run on the z/OS system where the specified DB2 subsystem is running. Ensure that the utility job runs on the appropriate z/OS system. You must use one of several z/OS installation-specific statements to make sure this happens. These include:

- For JES2 multi-access spool (MAS) systems, insert the following statement into the utility JCL:

  ```
  /*JOBPARM SYSAFF=cccc
  ```

- For JES3 systems, insert the following statement into the utility JCL:

  ```
  //*MAIN SYSTEM=(main-name)
  ```

The preceding JCL statements are described in *z/OS MVS JCL Reference*. Your installation might have other mechanisms for controlling where batch jobs run, such as by using job classes.

Stopping and restarting utilities: In a data sharing environment, you can terminate an active utility by using the TERM UTILITY command only on the DB2 subsystem on which it was started. If a DB2 subsystem fails while a utility is in progress, you must restart that DB2 subsystem, and then you can terminate the utility from any system.

You can restart a utility only on a member that is running the same DB2 release level as the member on which the utility job was originally submitted. The same utility ID (UID) must be used to restart the utility. That UID is unique within a

data sharing group. However, if DB2 fails, you must restart DB2 on either the same or another z/OS system before you restart the utility.

Terminating an online utility with the TERM UTILITY command

The information under this heading, up to "Restarting an online utility" on page 43 is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 905.

Use the TERM UTILITY command to terminate the execution of an active utility or to release the resources that are associated with a stopped utility.

Restriction: If the utility was started in a previous release of DB2, issue the TERM
UTILITY command from that release.

After you issue the TERM UTILITY command, you cannot restart the terminated utility job. The objects on which the utility was operating might be left in an indeterminate state. In many cases, you cannot rerun the same utility without first recovering the objects on which the utility was operating. The situation varies, depending on the utility and the phase that was in process when you issued the command. These considerations about the state of the object are particularly important when terminating the COPY, LOAD, and REORG utilities.

In a data sharing environment, TERM UTILITY is effective for active utilities when the command is submitted from the DB2 subsystem that originally issued the command. You can terminate a stopped utility from any active member of the data sharing group.

Restriction: In a data sharing coexistence environment, you can terminate a utility
only on the same release in which the utility was started.

If the utility is active, TERM UTILITY terminates it at the next commit point. It then performs any necessary cleanup operations.

You might choose to put TERM UTILITY in a conditionally executed job step; for example, if you never want to restart certain utility jobs. Figure 7 shows a sample job stream.

```
//TERM EXEC PGM=IKJEFT01,COND=((8,GT,S1),EVEN)
//*
//*************************************************************
//* IF THE PREVIOUS UTILITY STEP, S1, ABENDS, ISSUE A
//* TERMINATE COMMAND.  IT CANNOT BE RESTARTED.
//*************************************************************
//*
//SYSPRINT DD  SYSOUT=A
//SYSTSPRT DD  SYSOUT=A
//SYSOUT   DD  SYSOUT=A
//SYSUDUMP DD  SYSOUT=A
//SYSTSIN  DD  *
DSN SYSTEM(DSN)
-TERM UTILITY(TEMP)
END
/*
```

Figure 7. Example of conditionally executed TERM UTILITY

Alternatively, consider specifying the TIMEOUT TERM parameter for some Online REORG situations.

Restarting an online utility

If a utility finishes abnormally, you might be able to restart it. With the autonomic restart procedure, you avoid repeating much of the work that the utility has already done.

Before you restart a job, correct the problem that caused the utility job to stop. Then resubmit the job. DB2 recognizes the utility ID and restarts the utility job if possible. DB2 retrieves information about the stopped utility from the SYSUTIL directory table.

Do not reuse the utility ID of a stopped utility that has not yet been terminated, unless you want to restart that utility. If you do use the same utility ID to invoke a different utility, DB2 tries to restart the original stopped utility with the information that is stored in the SYSUTIL directory table.

Two different methods of restart are available:

- You can do a *phase restart* from the beginning of the phase that was being processed. This method is indicated by the value RESTART(PHASE).
- You can do a *current restart* from the last checkpoint that was taken during the execution of the utility phase. If the utility phase does not take any checkpoints or has not reached the first checkpoint, current restart is equivalent to phase restart. This method is indicated by the value RESTART or RESTART(CURRENT).

For each utility, DB2 uses the default RESTART value that is specified in Table 4. For a complete description of the restart behavior for an individual utility, including any phase restrictions, refer to the restart section for that utility.

You can override the default RESTART value by specifying the RESTART parameter in the original JCL data set. DB2 ignores the RESTART parameter if you are submitting the utility job for the first time. For instructions on how to specify this parameter, see "Using the RESTART parameter" on page 44.

Table 4. Default RESTART values for each utility

Utility	Default RESTART value
BACKUP SYSTEM	RESTART(CURRENT)
CATMAINT	No restart
CHECK DATA	RESTART(CURRENT)
CHECK INDEX	RESTART(CURRENT)
CHECK LOB	RESTART(CURRENT)
COPY	RESTART(CURRENT)
COPYTOCOPY	RESTART(CURRENT)
DIAGNOSE	Restarts from the beginning
EXEC SQL	Restarts from the beginning
LISTDEF	Restarts from the beginning
LOAD	RESTART(CURRENT) or RESTART(PHASE)[1]
MERGECOPY	RESTART(PHASE)
MODIFY RECOVERY	RESTART(CURRENT)
MODIFY STATISTICS	RESTART(CURRENT)

Table 4. Default RESTART values for each utility (continued)

Utility	Default RESTART value
OPTIONS	Restarts from the beginning
QUIESCE	RESTART(CURRENT)
REBUILD INDEX	RESTART(PHASE)
RECOVER	RESTART(CURRENT)
REORG INDEX	RESTART(CURRENT) or RESTART(PHASE)[1]
REORG TABLESPACE	RESTART(CURRENT) or RESTART(PHASE)[1]
REPAIR	No restart
REPORT	RESTART(CURRENT)
RESTORE SYSTEM	RESTART(CURRENT)
RUNSTATS	RESTART(CURRENT)
STOSPACE	RESTART(CURRENT)
TEMPLATE	Restarts from the beginning
UNLOAD	RESTART(CURRENT)

Notes:

1. The RESTART value that DB2 uses for these utilities depends on the situation. Refer to the restart section for each utility for a complete explanation.

If you cannot restart a utility job, you might have to terminate it to make the data available to other applications. To terminate a utility job, issue the DB2 TERM UTILITY command. Use the command only if you must start the utility from the beginning.

Using the RESTART parameter

You do not need to use the RESTART parameter to restart a utility job. When you resubmit a job that finished abnormally and has not been terminated, DB2 automatically recognizes the utility ID from the SYSUTIL directory table and restarts the utility job if possible. However, if you want to override the default RESTART value (as listed in Table 4 on page 43), you can update the original JCL data set by adding the RESTART parameter. Any RESTART values that you specify always override the default values. DB2 ignores the RESTART parameter if you are submitting the utility job for the first time.

To add the RESTART parameter, you can use one of the following three methods:

- **Using DB2I**. Add the RESTART parameter by following these steps:
  1. Access the DB2 Utilities panel.
  2. Fill in the panel fields, as documented in Figure 2 on page 25, except for field 5.
  3. Change field 5 to CURRENT or PHASE, depending on the desired method of restart.
  4. Press Enter.
- **Using the DSNU CLIST command**. When you invoke the DSNU CLIST command, as described in "Using the DSNU CLIST command in TSO" on page 27, change the value of the RESTART parameter by specifying either RESTART, RESTART (CURRENT), or RESTART(PHASE).

- **Creating your own JCL**. If you create your own JCL, you can specify RESTART (CURRENT) or RESTART(PHASE) to override the default RESTART value. You must also check the DISP parameters on the DD statements. For example, for DD statements that have DISP=NEW and need to be reused, change DISP to OLD or MOD. If generation data groups (GDGs) are used and any (+1) generations were cataloged, ensure that the JCL is changed to GDG (+0) for such data sets.

  Automatically generated JCL normally has DISP=MOD. DISP=MOD allows a data set to be allocated during the first execution and then reused during a restart.

  When restarting a job that involves templates, DB2 automatically changes the disposition from NEW to MOD. Therefore, you do not need to change template specifications for restart.

Adding or deleting utility statements

During restart processing, DB2 remembers the relative position of the stopped utility statement in the input stream. Therefore, you must include all the original utility statements when restarting any online utility; however, you can add or delete DIAGNOSE statements.

Modifying utility control statements

When restarting a utility job, do not change any EXEC SQL or OPTIONS utility control statements that have been executed prior to the stopped utility, if possible. If you must change these utility control statements, use caution; any changes can cause the restart processing to fail. For example, if you specify a valid OPTIONS statement in the initial invocation, and then on restart, specify OPTIONS PREVIEW, the restart fails.

Use caution when changing LISTDEF lists prior to a restart. When DB2 restarts list processing, it uses a saved copy of the list. Modifying the LISTDEF list that is referred to by the stopped utility has no effect. Only control statements that follow the stopped utility are affected.

Do not change the position of any other utilities that have been executed.

Restarting after the output data set is full

If a utility job terminates with an out-of-space condition on the output data set and you want to restart the job at the last commit point, follow these steps:

1. Copy the output data set to a temporary data set. Use the same DCB parameters. Use z/OS utilities that do not reblock the data set during the copy operation (for example, DFDSS ADRDSSU or DFSORT ICEGENER). Avoid using the IEBGENER or ISPF 3.3 utilities.
2. Delete or rename the output data set. Ensure that you know the current DCB parameters, and then redefine the data set with additional space. Use the same VOLSER (if the data set is not cataloged), the same DSNAME, and the same DCB parameters.
3. Copy the data from the temporary data set into the new, larger output data set. Use z/OS utilities that do not reblock the data set during the copy operation (for example, DFDSS ADRDSSU or DFSORT ICEGENER).

Restarting with templates

Unlike most other utility control statements, TEMPLATE control statements can be modified before restarting a utility, and, in some cases, they must be modified in

order to correct a prior failure. However, use caution when modifying templates. In some cases, modifications can cause restart processing to fail. For example, if you change the template name of a temporary work data set that was opened in an earlier phase and closed but is to be used later, restart processing fails.

TEMPLATE allocation during a restart automatically adjusts data set dispositions to reallocate the data sets from the prior execution. No modification to the TEMPLATE DISP is required. If the prior failure was due to space problems on a data set, the same restart considerations apply as if DD statements were being used. If the prior failure was due to insufficient space on a volume, you can alter the TEMPLATE statement. How the TEMPLATE statement needs to be altered depends on whether the SPACE keyword was specified. If SPACE was specified, specify a different volume or alter the primary and secondary space quantities. If SPACE was not specified, specify a different volume or add the PCTPRIME and NBRSECND keywords. Lower PCTPRIME to decrease the size of the primary allocation, and increase NBRSECND to decrease the size of the secondary allocation. DB2 takes checkpoints for the values that are used for TEMPLATE DSN variables, and the old values are reused on restart.

Restarting with lists

Unlike other utility control statements, LISTDEF control statements can be modified before restarting a utility. However, the modification does not affect the currently running utility. It affects only those utilities that follow it.

If the utility that you are restarting was processing a LIST, you will see a list size that is greater than 1 on the DSNU100 or DSNU105 message. DB2 checkpoints the expanded, enumerated list contents prior to executing the utility. DB2 uses this checkpointed list to restart the utility at the point of failure. After a successful restart, the LISTDEF is re-expanded before subsequent utilities in the same job step use it.

Other restart hints

The following guidelines provide additional information about restarting utilities:

- If the data set is not dynamically allocated, ensure that the DD name that is specified in the restart JCL matches the DD name for the original job; don't change DD names on a restart job. If the data set is dynamically allocated, the file sequence numbers must match for the restart and the original run. In either case, if the data set is not cataloged, any explicit specification of VOLSERs must match for the restart and the original job. If you copy a work data set, such as SYSUT1, after an ABENDB37, and the number of volumes changes, do not specify RESTART CURRENT. If you do, ABEND 413-1C occurs. To prevent this abend, start the utility in RESTART(PHASE).
- When restarting a utility with cataloged data sets, do not specify VOLSER. Let DB2 determine the VOLSER of the data sets from the system catalog.
- Based on the specific scenario in a RESTART job, the processing is different when using TAPE STACK(YES). If the data set has not been allocated in the first invocation of the job, the remaining inline copy data sets will be copied to a new tape volume with the file sequence numbers starting at one. When a mount request is issued for a private tape, ensure that a new volume is mounted so that the existing files on tape are not overwritten. You can use the expiration date to enforce this process. If the data set has already been allocated in the first invocation of the job, the remaining inline copy data sets will be copied to the tape volume that was used for the initial job with the file sequence numbers continuing from the setting that was valid when the initial job was terminated

abnormally. In this case, the mount requests the VOLSER from the original job and does not ask for a PRIVATE volume. This is consistent with how restart works for the COPY utility when the output image copies are stacked on tape.

- Do not change the utility function that is currently stopped and the DB2 objects on which it is operating. However, you can change other parameters that are related to the stopped step and to subsequent utility steps.
- Do not specify z/OS automatic step restart.
- If a utility is restarted in the UTILINIT phase, it is re-executed from the beginning of the phase.
- Run the RUNSTATS utility after the completion of a restarted LOAD, REBUILD INDEX, or REORG job with the STATISTICS option. When you restart these jobs, DB2 does not collect inline statistics. The exception is REORG UNLOAD PAUSE; when restarted after the pause, REORG UNLOAD PAUSE collects statistics.
- Ensure that the required data sets are properly defined. The recommended data dispositions for data sets on RESTART are listed in Table 125 on page 602. **Recommendation:** Allocate the data sets by using TEMPLATE statements that do not specify the DISP and SPACE parameter values. When these parameters are not specified, DB2 determines the correct disposition and size of these data sets.
- When using the DSNUTILS stored procedure, specify NONE or ANY for the *utility-name* parameter. These values suppress the dynamic allocation that is normally performed by DSNUTILS. You can then specify TEMPLATE statements (in the *utstmt* parameter) to allocate the necessary data sets.

Restart is not always possible. The restrictions applying to the phases of each utility are discussed under the description of each utility.

Chapter 5. BACKUP SYSTEM

The online BACKUP SYSTEM utility invokes z/OS DFSMShsm (Version 1 Release 5 or above) to copy the volumes on which the DB2 data and log information resides for either a DB2 subsystem or data sharing group. You can use BACKUP SYSTEM to copy all data for a single application (for example, when DB2 is the database server for a resource planning solution). All data sets that you want to copy must be SMS-managed data sets. You can subsequently run the RESTORE SYSTEM utility to recover the data.

In a data sharing environment, if any failed or abnormally quiesced members exist, the BACKUP SYSTEM request fails.

The BACKUP SYSTEM utility uses copy pools, which are new constructs in z/OS DFSMShsm V1R5. A *copy pool* is a defined set of storage groups that contain data that DFSMShsm can backup and recover collectively. For more information about copy pools, see *z/OS DFSMSdfp Storage Administration Reference*.

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.

Output: The output for BACKUP SYSTEM is the copy of the volumes on which the DB2 data and log information resides. The BACKUP SYSTEM history is recorded in the bootstrap data sets (BSDSs).

Related information: For information about the use of BACKUP SYSTEM in point-of-time recovery, see Part 4 of *DB2 Administration Guide*.

Authorization required: To execute this utility, you must use a privilege set that includes SYSCTRL or SYSADM authority.

Execution phases of BACKUP SYSTEM: The BACKUP SYSTEM utility operates in these phases:

Phase	Description
UTILINIT	Performs initialization and setup
COPY	Copies data
UTILTERM	Performs cleanup

The following topics provide additional information:

- "Syntax and options of the BACKUP SYSTEM control statement " on page 50
- "Instructions for running BACKUP SYSTEM" on page 50
- "Concurrency and compatibility for BACKUP SYSTEM" on page 52
- "Sample BACKUP SYSTEM control statements" on page 53

Syntax and options of the BACKUP SYSTEM control statement

The utility control statement defines the function that the utility job performs. Use the ISPF/PDF edit function to create a control statement and to save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

When you specify BACKUP SYSTEM, you can specify only the following statements in the same step:

- DIAGNOSE
- OPTIONS PREVIEW
- OPTIONS OFF
- OPTIONS KEY
- OPTIONS EVENT WARNING

In addition, BACKUP SYSTEM must be the last statement in SYSIN.

Syntax diagram

Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

FULL
Indicates that you want to copy both the database copy pool and the log copy pool. The **default** is **FULL**.

You must ensure that the database copy pool is set up to contain the volumes for the databases and the associated integrated catalog facility (ICF) catalogs. You must also ensure that the log copy pool is set up to contain the volumes for the BSDSs, the active logs, and the associated catalogs.

Use BACKUP SYSTEM FULL to allow for recovery of both data and logs. You can use the RESTORE SYSTEM utility to recover the data. However, RESTORE SYSTEM does not restore the logs; the utility only applies the logs. If you want to restore the logs, you must use another method to restore them.

DATA ONLY
Indicates that you want to copy only the database copy pool. You must ensure that the database copy pool is set up to contain the volumes for the databases and the associated ICF catalogs.

Instructions for running BACKUP SYSTEM

To run BACKUP SYSTEM, you must:

1. Read "Before running BACKUP SYSTEM" on page 51.

2. Prepare the necessary data sets, as described in "Data sets that BACKUP SYSTEM uses."
3. Create JCL statements by using one of the methods that are described in either "Using the supplied JCL procedure (DSNUPROC)" on page 34 or "Creating the JCL data set yourself by using the EXEC statement" on page 37.
4. Prepare a utility control statement that specifies the options for the tasks that you want to perform.
5. Check "Concurrency and compatibility for BACKUP SYSTEM" on page 52 if you want to run other jobs concurrently on the same target objects.
6. Plan for restarting BACKUP SYSTEM if the job doesn't complete, as described in "Terminating or restarting BACKUP SYSTEM" on page 52.
7. Run BACKUP SYSTEM by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Before running BACKUP SYSTEM

To run BACKUP SYSTEM, 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.
- The ICF catalog for the data is on a separate volume than the ICF catalog for the logs.
- You have defined an SMS backup storage group for each storage group in the copy pools.

For information about defining copy pools and associated backup storage groups, see *z/OS DFSMSdfp Storage Administration Reference*. Use the following DB2 naming convention when you define these copy pools:

```
DSN$locn-name$cp-type
```

The variables that are used in this naming convention have the following meanings:

DSN The unique DB2 product identifier.

**$** A delimiter. You must use the dollar sign character ($).

locn-name
The DB2 location name.

cp-type The copy pool type. Use DB for database and LG for log.

Data sets that BACKUP SYSTEM uses

Table 5 on page 52 lists the data sets that the BACKUP SYSTEM utility uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set

Table 5. Data sets that BACKUP SYSTEM uses

Data sets	Description	Required?
SYSIN	An input data set that contains the utility control statement	Yes
SYSPRINT	An output data set for messages	Yes

Creating the control statement

Create the utility control statement for the BACKUP SYSTEM job. See "Syntax diagram" on page 61 for BACKUP SYSTEM syntax and option descriptions. See "Sample BACKUP SYSTEM control statements" on page 53 for examples of BACKUP SYSTEM usage.

Instructions for specific tasks

To use the DISPLAY UTILITY command for BACKUP SYSTEM on a data sharing group, you must issue the command from the member on which the BACKUP SYSTEM utility is invoked. Otherwise, the current utility information is not displayed.

Terminating or restarting BACKUP SYSTEM

You can terminate BACKUP SYSTEM by using the TERM UTILITY command. BACKUP SYSTEM checks for the TERM UTILITY command before the call to copy data. TERM UTILITY deletes the copy that is being created through the BACKUP SYSTEM utility.

To use TERM UTILITY to terminate BACKUP SYSTEM on a data sharing group, you must issue the command from the member on which the BACKUP SYSTEM utility is invoked.

You can restart a BACKUP SYSTEM utility job, but it starts from the beginning again. For guidance in restarting online utilities, see "Restarting an online utility" on page 43.

Concurrency and compatibility for BACKUP SYSTEM

BACKUP SYSTEM can run concurrently with any other utility; however, it must wait for the following DB2 events to complete before the copy can begin:

- Extending of data sets
- Writing of 32-KB pages
- Writing close page set control log records (PSCRs)
- Creating data sets (for table spaces, indexes, and so forth)
- Deleting data sets (for dropping tables spaces, indexes, and so forth)
- Renaming data sets (for online reorganizing of table spaces, indexes, and so forth during the SWITCH phase)

Only one BACKUP SYSTEM job can be running at one time.

Sample BACKUP SYSTEM control statements

Example 1: Creating a full backup of a DB2 subsystem or data sharing group. The following control statement specifies that the BACKUP SYSTEM utility is to create a full backup copy of a DB2 subsystem or data sharing group. The full backup includes copies of both the database copy pool and the log copy pool. In this control statement, the FULL option is not explicitly specified, because it is the default.

```
//STEP1    EXEC DSNUPROC,TIME=1440,
//         UTPROC='',
//         SYSTEM='DSN'
//SYSIN    DD *
  BACKUP  SYSTEM
/*
```

Example 2: Creating a data-only backup of a DB2 subsystem or data sharing group. The following control statement specifies that BACKUP SYSTEM is to create a backup copy of only the database copy pool for a DB2 subsystem or data sharing group.

```
//STEP1    EXEC DSNUPROC,TIME=1440,
//         UTPROC='',
//         SYSTEM='DSN'
//SYSIN    DD *
  BACKUP  SYSTEM  DATA ONLY
/*
```

Chapter 6. CATENFM

The CATENFM utility enables a DB2 subsystem to enter DB2 Version 8 enabling-new-function mode and Version 8 new-function mode. All new Version 8 functions are unavailable until the subsystem enters new-function mode. For more information about CATENFM, including instructions for running it, see *DB2 Installation Guide*.

Chapter 7. CATMAINT

The CATMAINT utility updates the catalog; run this utility during migration to a new release of DB2 or when IBM Software Support instructs you to do so. For more information about CATMAINT, including instructions for running it, see *DB2 Installation Guide*.

Chapter 8. CHECK DATA

The CHECK DATA utility checks table spaces for violations of referential and table check constraints, and reports information about violations that it detects. CHECK DATA also checks for consistency between a base table space and the corresponding LOB table space. The utility does not check informational referential constraints.

Run CHECK DATA after a conditional restart or a point-in-time recovery on all table spaces where parent and dependent tables might not be synchronized or where base tables and auxiliary tables might not be synchronized. You can run CHECK DATA against a base table space only, not against a LOB table space.

Restriction: Do not run CHECK DATA on encrypted data. Because CHECK DATA does not decrypt the data, the utility might produce unpredictable results.

For a diagram of CHECK DATA syntax and a description of available options, see "Syntax and options of the CHECK DATA control statement" on page 60. For detailed guidance on running this utility, see "Instructions for running CHECK DATA" on page 65.

Output: CHECK DATA optionally deletes rows that violate referential or table check constraints. CHECK DATA copies each row that violates one or more constraints to an exception table. If a row violates two or more constraints, the row is copied only once.

If the utility finds any violation of constraints, CHECK DATA puts the table space that it is checking in the CHECK-pending status.

You can specify that the CHECK-pending status is to be reset when CHECK DATA execution completes.

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:
- STATS privilege for the database
- DBADM, DBCTRL, or DBMAINT authority for the database
- SYSCTRL or SYSADM authority

An ID with installation SYSOPR authority can also execute CHECK DATA. However, you cannot use SYSOPR authority to execute CHECK DATA on table space SYSDBASE in database DSNDB06 or on any object except SYSUTILX in database DSNDB01.

If you specify the DELETE option, the privilege set must include the DELETE privilege on the tables that are being checked. If you specify the FOR EXCEPTION option, the privilege set must include the INSERT privilege on any exception table that is used. If you specify the AUXERROR INVALIDATE option, the privilege set must include the UPDATE privilege on the base tables that contain LOB columns.

Execution phases of CHECK DATA:

Phase	Description
UTILINIT	Performs initialization

SCANTAB	Extracts foreign keys; uses foreign key index if it matches exactly; otherwise scans the table
SORT	Sorts foreign keys if they are not extracted from the foreign key index
CHECKDAT	Looks in primary indexes for foreign key parents, and issue messages to report detected errors
REPORTCK	Copies error rows into exception tables, and delete them from source table if DELETE YES is specified
UTILTERM	Performs cleanup

The following topics provide additional information:

Syntax and options of the CHECK DATA control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram

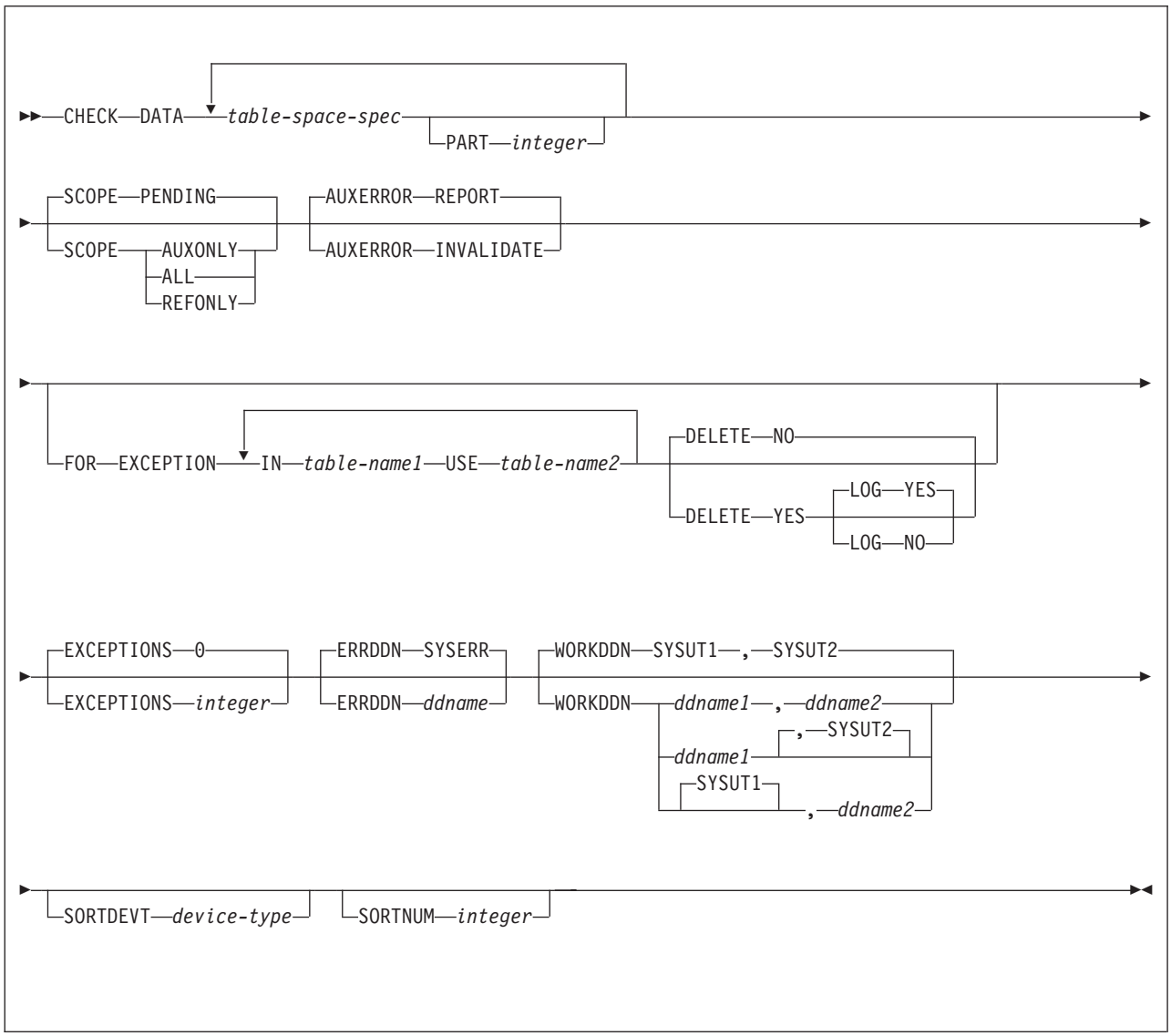
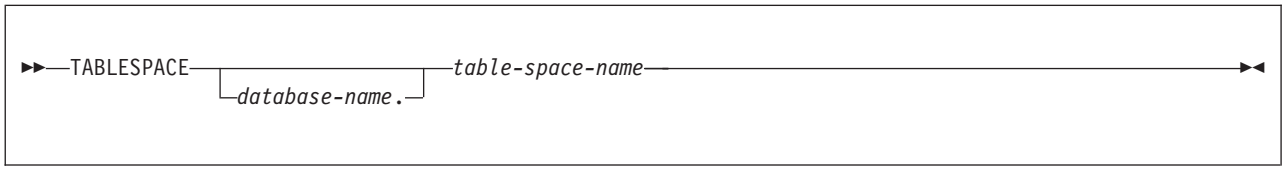


table-space-spec:



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

DATA Indicates that you want the utility to check referential and table check constraints. CHECK DATA does not check informational referential constraints.

TABLESPACE *database-name.table-space-name*
Specifies the table space to which the data belongs.

database-name is the name of the database and is optional. The **default** is **DSNDB04**.

table-space-name is the name of the table space.

PART *integer* Identifies which partition to check for constraint violations.

integer is the number of the partition and must be in the range from 1 to the number of partitions that are defined for the table space. The maximum is 4096.

SCOPE Limits the scope of the rows in the table space that are to be checked.

PENDING
Indicates that the only rows that are to be checked are those that are in table spaces, partitions, or tables that are in CHECK-pending status. The referential integrity check, constraint check, and the LOB check are all performed.

If you specify this option for a table space that is **not** in CHECK-pending status, the CHECK DATA utility does not check the table space and does not issue an error message.

The **default** is **PENDING**.

AUXONLY
Indicates that only the LOB column check is to be performed for table spaces that have tables with LOB columns. The referential integrity and constraint checks are not performed.

ALL Indicates that all dependent tables in the specified table spaces are to be checked. The referential integrity check, constraint check, and the LOB check are performed.

REFONLY
Same as the **ALL** option, except that the LOB column check is not to be performed.

AUXERROR Specifies the action that CHECK DATA is to perform when it finds a LOB column check error.

REPORT A LOB column check error is reported with a warning message. The base table space is set to the auxiliary CHECK-pending (ACHKP) status.

The **default** is **REPORT**.

INVALIDATE A LOB column check error is reported with a warning message. The base table LOB column is set to an invalid status. A LOB column with invalid status that is now correct is set valid. This action is also reported with a message. The base table space is set to the auxiliary warning (AUXW) status if any LOB column remains in invalid status.

Before using CHECK DATA to check LOBs:

1. Run CHECK LOB to ensure the validity of the LOB table space.

2. Run REBUILD INDEX or CHECK INDEX on the index on the auxiliary table to ensure its validity.

FOR EXCEPTION

Indicates that any row that is in violation of referential or table check constraints is to be copied to an exception table. Although this keyword does not apply to the checking of LOB columns, rows with LOB columns are moved to the exception tables. If you specify AUXONLY for LOB checking only, the FOR EXCEPTION option is ignored.

If any row violates more than one constraint, that row appears no more than once in the exception table.

IN *table-name1*

Specifies the table (in the table space that is specified on the TABLESPACE keyword) from which rows are to be copied.

table-name1 is the name of the table. Enclose the table name in quotation marks if the name contains a blank.

USE *table-name2*

Specifies the exception table into which error rows are to be copied.

table-name2 is the name of the exception table and must be a base table; it cannot be a view, synonym, or alias. Enclose the table name in quotation marks if the name contains a blank.

DELETE

Indicates whether rows that are in violation of referential or table check constraints are to be deleted from the table space. You can use this option only if you specify the FOR EXCEPTION keyword.

NO Indicates that error rows are to remain in the table space. Primary errors in dependent tables are copied to exception tables. The **default** is **NO**.

If DELETE NO is specified, and constraint violations are detected, CHECK DATA places the table space in the CHECK-pending status.

YES Indicates that error rows are to be deleted from the table space. Deleted rows from both dependent and descendent tables are placed into exception tables.

LOG

Specifies the logging action that is to be taken when records are deleted.

YES

Logs all records that are deleted during the REPORTCK PHASE.

NO

Does not log any records that are deleted during the REPORTCK phase. If any rows are deleted, CHECK DATA places the table space in the COPY-pending status, and it places any indexes that were defined with the COPY YES attribute in the informational COPY-pending status.

Attention: Use the LOG NO option with caution because its use limits your ability to recover data by using the log. For example, if you issue the CHECK DATA DELETE YES LOG NO utility control statement at particular log RBA, you can recover data that exists on the lob before that point in time or after the point on the log at which the utility execution completes.

EXCEPTIONS *integer*

Specifies the maximum number of exceptions, which are reported by messages only. CHECK DATA terminates in the CHECKDAT phase when it reaches the specified number of exceptions; if termination occurs, the error rows are not written to the EXCEPTION table.

Only records that contain primary referential integrity errors or table check constraint violations are applied toward the exception limit. The number of records that contain secondary errors is not limited.

integer is the maximum number of exceptions. The **default** is **0**, which indicates no limit on the number of exceptions.

ERRDDN *ddname*

Specifies a DD statement for an error processing data set.

ddname is either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593. The **default** is **SYSERR.**

WORKDDN (*ddname1***,***ddname2***)**

Specifies the DD statements for the temporary work file for sort input and the temporary work file for sort output. A temporary work file for sort input and output is required.

You can use the WORKDDN keyword to specify either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, WORKDDN uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

ddname1 is the DD name of the temporary work file for sort input. The **default** is **SYSUT1**.

ddname2 is the DD name of the temporary work file for sort output. The **default** is **SORTOUT**.

SORTDEVT *device-type*

Specifies the device type for temporary data sets that are to be dynamically allocated by DFSORT. *device-type* is the device type can be any device type that is acceptable to the DYNALLOC parameter of the SORT or OPTION control statement for DFSORT, as described in *DFSORT Application Programming: Guide.*

Do not use a TEMPLATE specification to dynamically allocate sort work data sets. The presence of the SORTDEVT keyword controls dynamic allocation of these data sets.

SORTNUM *integer*
Specifies the number of temporary data sets that are to be dynamically allocated by the sort program.

integer is the number of temporary data sets that can range from 2 to 255.

If you omit SORTDEVT, SORTNUM is ignored. If you use SORTDEVT and omit SORTNUM, no value is passed to DFSORT; DFSORT uses its own default.

You need at least two sort work data sets for each sort. The
SORTNUM value applies to each sort invocation in the utility. For
example, if there are three indexes, SORTKEYS is specified, there
are no constraints limiting parallelism, and SORTNUM is specified
as 8, then a total of 24 sort work data sets will be allocated for a
job.

Each sort work data set consumes both above the line and below
the link virtual storage, so if you specify too high a value for
SORTNUM, the utility may decrease the degree of parallelism due
to virtual storage constraints, and possibly decreasing the degree
down to one, meaning no parallelism.

Important: The SORTNUM keyword will not be considered if
ZPARM UTSORTAL is set to YES and IGNSORTN is
set to YES.

Instructions for running CHECK DATA

To run CHECK DATA, you must:

1. Read "Before running CHECK DATA."
2. Prepare the necessary data sets, as described in "Data sets that CHECK DATA uses" on page 69.
3. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for CHECK DATA, see "Sample CHECK DATA control statements" on page 75.)
4. Prepare a utility control statement that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks" on page 70.
5. Check the compatibility table in "Concurrency and compatibility for CHECK DATA" on page 74 if you want to run other jobs concurrently on the same target objects.
6. Plan for restarting CHECK DATA if the job doesn't complete, as described in "Terminating or restarting CHECK DATA" on page 73.
7. Run CHECK DATA by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Before running CHECK DATA

This section describes the actions that you must take before you can run the CHECK DATA utility.

For a table with no LOB columns

Before running CHECK DATA, you should run CHECK INDEX on primary key indexes and foreign key indexes to ensure that the indexes that CHECK DATA uses are valid. This action is especially important before using CHECK DATA with the DELETE YES or PART options.

For a table with LOB columns

If you plan to run CHECK DATA on a base table space that contains at least one LOB column, complete the following steps prior to running CHECK DATA:

1. Run CHECK LOB on the LOB table space.
2. Run CHECK INDEX on the index on the auxiliary table to ensure the validity of the LOB table space and the index on the auxiliary table.
3. Run CHECK INDEX on the indexes on the base table space.

The relationship between a base table with a LOB column and the LOB table space is shown in Figure 8. The LOB column in the base table points to the auxiliary index on the LOB table space, as illustrated in the figure. For more information about LOBs and auxiliary tables, see Part 2 of *DB2 Administration Guide*.

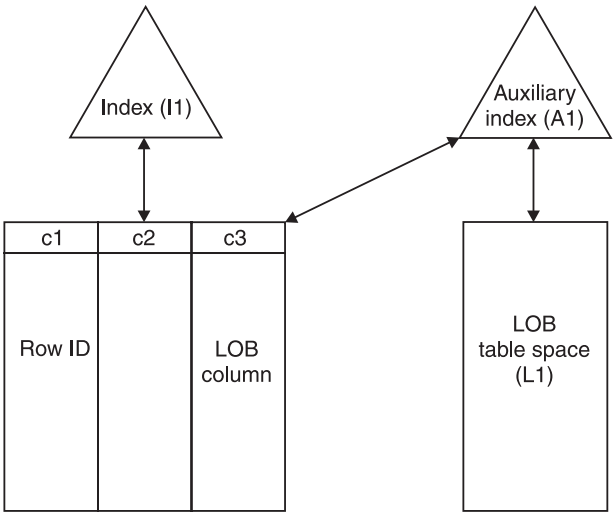


Figure 8. Relationship between a base table with a LOB column and the LOB table space

If the LOB table space is in either the CHECK-pending or RECOVER-pending status, or if the index on the auxiliary table is in REBUILD-pending status, CHECK DATA issues an error message and fails.

Create exception tables

An *exception table* is a user-created table that duplicates the definition of a dependent table. The CHECK DATA utility checks the dependent table, which consists of at least *n* columns, where *n* is the number of columns of the dependent table. The CHECK DATA utility copies the deleted rows from the dependent table to the exception table. Table 6 describes the contents of an exception table. This table lists the columns, a description of the column content, whether or not the column is required, the data type and length of the column value, and whether or not the column has the NULL attribute.

Table 6. Contents of exception tables

Column	Description	Required?	Data type and length	NULL attribute
1 to *n*	Corresponds to columns in the table that is being checked. These columns hold data from table rows that violate referential or table check constraints.	Yes	The same as the corresponding columns in the table that is being checked.	The same as the corresponding columns in the table that is being checked.
n+1	Identifies the RIDs of the invalid rows of the table that is being checked.	No	CHAR(4); CHAR(5)[1] for table spaces that are defined with LARGE or DSSIZE options	Anything
n+2	Starting time of the CHECK DATA utility.	No	TIMESTAMP	Anything
≥ *n*+2	Additional columns that the CHECK DATA utility does not use.	No	Anything	Anything

Notes:

1. You can use CHAR(5) for any type of table space, but you must use it for table spaces that are defined with the LARGE or DSSIZE options.

If you delete rows by using the CHECK DATA utility with SCOPE ALL, you must create exception tables for all tables that are named in the table spaces and for all their descendents. All descendents of any row are deleted.

When creating or using exception tables, be aware of the following guidelines:

- The exception tables should not have any unique indexes or referential or table check constraints that might cause errors when CHECK DATA inserts rows into them.
- You can create a new exception table before you run CHECK DATA, or you can use an existing exception table. The exception table can contain rows from multiple invocations of CHECK DATA.
- If column *n*+2 is of type TIMESTAMP, CHECK DATA records the starting time. Otherwise, it does not use column *n*+2.
- You must have DELETE authorization on the dependent table that is being checked.
- You must have INSERT authorization on the exception table.
- Column names in the exception table can have any name.
- Any change to the structure of the dependent table (such as a dropped column) is not automatically recorded in the exception table. You must make that change in the exception table.

Exception processing a table with a LOB column

If you use exception tables, the exception table for the base table must have a similar LOB column and a LOB table space for each LOB column. If an exception is found, DB2 moves the base table row with its LOB column to the exception table. Then DB2 moves the LOB column into the exception table's LOB table space. If you specify DELETE YES, DB2 deletes the base table row and the LOB column.

An auxiliary table cannot be an exception table. A LOB column check error is not included in the exception count. A row with only a LOB column check error does not participate in exception processing.

Example: creating an exception table for the project activity table

General-use Programming Interface

You can create an exception table for the project activity table by using the following SQL statements:

```
EXEC SQL
CREATE TABLE EPROJACT
   LIKE DSN8810.PROJACT
   IN DATABASE DSN8D81AENDEXEC

EXEC SQL
ALTER TABLE EPROJACT
   ADD RID CHAR(4)
ENDEXEC

EXEC SQL
ALTER TABLE EPROJACT
   ADD TIME TIMESTAMP NOT NULL WITH DEFAULT
ENDEXEC
```

The first statement requires the SELECT privilege on table DSN8810.PROJACT and the privileges that are usually required to create a table.

Table EPROJACT has the same structure as table DSN8810.PROJACT, but it can have two extra columns. The columns in EPROJACT are:

- Its first five columns mimic the columns of the project activity table; they have exactly the same names and descriptions. Although the column names are the same, they do not need to be. However, the rest of the column attributes for the initial columns must be same as those of the table that is being checked.
- The next column, which is added by ALTER TABLE, is optional; CHECK DATA uses it as an identifier. The name “RID” is an arbitrary choice; if the table already has a column with that name, use a different name. The column description, CHAR(4), is required.
- The final timestamp column is also optional. If you define the timestamp column, a row identifier (RID) column must precede this column. You might define a permanent exception table for each table that is subject to referential or table check constraints. You can define it once and use it to hold invalid rows that CHECK DATA detects. The TIME column allows you to identify rows that were added by the most recent run of the utility.

Eventually, you correct the data in the exception tables, perhaps with an SQL UPDATE statement, and transfer the corrections to the original tables by using statements that are similar to those in the following example:

```
INSERT INTO DSN8810.PROJACT
   SELECT PROJNO, ACTNO, ACSTAFF, ACSTDATE, ACENDATE
     FROM EPROJACT
     WHERE TIME > CURRENT TIMESTAMP - 1 DAY;
```

End of General-use Programming Interface

Complete all LOB column definitions

You must complete all LOB column definitions for a base table before running CHECK DATA. A LOB column definition is not complete until the LOB table space, auxiliary table, and index on the auxiliary table have been created. If any LOB column definition is not complete, CHECK DATA fails and issues error message DSNU075E.

Data sets that CHECK DATA uses

Table 7 lists the data sets that CHECK DATA uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 7. Data sets that CHECK DATA uses

Data set	Description	Required?
SYSIN	An input data set that contains the utility control statement.	Yes
SYSPRINT	An output data set for messages.	Yes
Work data sets	Two temporary data sets for sort input and sort output. Specify the DD names by using the WORKDDN option of the utility control statement. The default *ddname* for sort input is SYSUT1. The default *ddname* for sort output is SORTOUT. To find the approximate size in bytes of the work data sets, see"Defining work data sets" on page 69.	Yes
Error data set	An output data set that collects information about violations that are encountered during the CHECKDAT phase for referential constraints or the SCANTAB phase for check constraints. Specify the DD name by using the ERRDDN parameter of the utility control statement. The default *ddname* is SYSERR.	Yes
UTPRINT	A data set that contains messages from DFSORT (usually, SYSOUT or DUMMY).	No

The following objects are named in the utility control statement and do not require DD statements in the JCL:

Table space
Object that is to be checked. (If you want to check only one partition of a table space, use the PART option in the control statement.)

Exception table
Table that stores rows that violate any referential constraints. For each table in a table space that is checked, specify the name of an exception table in the utility control statement. Any row that violates a referential constraint is copied to the exception table.

Defining work data sets: Three sequential data sets are required during execution of CHECK DATA. Two work data sets and one error data set are described by DD statements in the WORKDDN and ERRDDN options.

To find the approximate size, in bytes, of the WORKDDN data set:

1. If a table space has a LOB column, count a total of 70 bytes for the LOB column, and then go to step 3. If a table space does not have a LOB column, go to step 2.
2. Add 18 to the length of the longest foreign key. For nonpadded indexes, the length of the longest foreign key is the maximum possible length of the key with all varying-length columns in the key padded to their maximum length, plus 2 bytes for each varying-length column.
3. Multiply the sum by the number of keys and LOB columns that are checked.

Create the ERRDDN data set so that it is large enough to accommodate one error entry (length=60 bytes) per violation that CHECK DATA detects.

DB2 utilities uses DFSORT to perform sorts. Sort work data sets cannot span volumes. Smaller volumes require more sort work data sets to sort the same amount of data; therefore, large volume sizes can reduce the number of needed sort work data sets. It is recommended that at least 1.2 times the amount of data to be sorted be provided in sort work data sets on disk. For more information about DFSORT, see *DFSORT Application Programming Guide*.

Creating the control statement

Create the utility control statement for the CHECK DATA job. See "Syntax diagram" on page 61 for CHECK DATA syntax and option descriptions. See "Sample CHECK DATA control statements" on page 75 for examples of CHECK DATA usage.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in your utility control statement:

Specifying the scope of CHECK DATA

Running CHECK DATA with SCOPE PENDING is normally sufficient. DB2 records which data rows must be checked to ensure the referential integrity of the table space.

Whenever the scope information is in doubt, run the utility with the SCOPE ALL option. The scope information is recorded in the DB2 catalog. The scope information can become indoubt whenever you start the target table space with ACCESS(FORCE), or when the catalog is recovered to a point in time.

If you want to check only the tables with LOB columns, specify the AUXONLY option. If you want to check all dependent tables in the specified table spaces **except** tables with LOB columns, specify the REFONLY option.

Checking several table spaces

To check several table spaces, you can specify more than one table space in a CHECK DATA control statement. This technique is useful for checking a complete

set of referentially related table spaces. The following example shows a CHECK DATA control statement that lists more than one table space.

```
CHECK DATA
    TABLESPACE DBJM1203.TLJM1203
    TABLESPACE DBJM1203.TPJM1204
      FOR EXCEPTION IN TLJM1203.TBJM1203 USE ADMF001.EXCPT3
           IN TPJM1204.TMBJM1204 USE ADMF001.EXCPT4
    DELETE YES
```

Finding violations

CHECK DATA issues a message for every row that contains a referential or table check constraint violation. The violation is identified by:

- The RID of the row
- The name of the table that contains the row
- The name of the constraint that is being violated

Figure 9 shows an example of messages that CHECK DATA issues.

```
DSNU0501   DSNUGUTC - CHECK DATA TABLESPACE DBJM1203.TLJM1203
            TABLESPACE DBJM1203.TPJM1204
            FOR EXCEPTION IN TLJM1203.TBJM1203 USE ADMF001.EXCPT3
            IN TPJM1204.TBJM1204 USE ADMF001.EXCPT4 DELETE YES
DSNU7271 = DSNUKINP - TABLESPACE 'DBJM1203.TLJM1203' IS NOT CHECK PENDING

DSNU7301   DSNUKDST - CHECKING TABLE TPJM1204.TBJM1204
DSNU0421   DSNUGSOR - SORT PHASE STATISTICS -
            NUMBER OF RECORDS=4
            ELAPSED TIME=00:00:00
DSN0733I   DSNUKERK - ROW (RID=X'000000020B') HAS NO PARENT FOR
TPJM1204.TBJM1204.TABFK
DSN0733I   DSNUKERK - ROW (RID=X'0010000201') HAS NO PARENT FOR
TPJM1204.TBJM1204.TABFK
DSN0733I   DSNUKERK - ROW (RID=X'002000020B') HAS NO PARENT FOR
TPJM1204.TBJM1204.TABFK
DSN0733I   DSNUKERK - ROW (RID=X'0030000201') HAS NO PARENT FOR
TPJM1204.TBJM1204.TABFK
DSNU739I   DSNUKDAT - CHECK TABLE TPJM1204.TBJM1204 COMPLETE, ELAPSED
TIME=00:00:00
DSNU741I = DSNUKRDY - 4 ROWS DELETED FROM TABLE TPJM1204.TBJM1204
DSNU568I = DSNUGSRX - INDEX TPJM1204.IPJM1204 IS IN INFORMATIONAL COPY PENDING
DSNU568I = DSNUGSRX - INDEX TPJM1204.IXJM1204 IS IN INFORMATIONAL COPY PENDING
DSNU749I   DSNUK001 - CHECK DATA COMPLETE,ELAPSED TIME=00:00:02
DSNU010I   DSNUGBAC - UTILTIY EXECTUION COMPLETE, HIGHEST RETURN CODE=4
```

Figure 9. Example of messages that CHECK DATA issues

Detecting and correcting constraint violations

To avoid problems, you should run CHECK DATA with DELETE NO to detect the violations before you attempt to correct the errors. If required, use DELETE YES after you analyze the output and understand the errors.

You can automatically delete rows that violate referential or table check constraints by specifying CHECK DATA with DELETE YES. However, you should be aware of the following possible problems:

- The violation might be created by a non-referential integrity error. For example, the indexes on a table might be inconsistent with the data in a table.
- Deleting a row might cause a cascade of secondary deletes in dependent tables. The cascade of deletes might be especially inconvenient within referential integrity cycles.
- The error might be in the parent table.

CHECK DATA uses the primary key index and all indexes that exactly match a foreign key. Therefore, before running CHECK DATA, ensure that the indexes are consistent with the data by using the CHECK INDEX utility.

Resetting CHECK-pending status

Take one of the following actions to remove CHECK-pending status:

- Use the DELETE NO option if no tables contain rows that violate referential or table check constraints.
- Use the DELETE YES option to remove all rows that violate referential or table check constraints.

If you run CHECK DATA with the DELETE NO option and referential or table check constraint violations are found, the table space or partition is placed in CHECK-pending status.

Interpreting LOB column errors

If you run CHECK DATA AUXERROR REPORT or INVALIDATE on a base table space that contains at least one LOB column, the following errors might be reported:

Orphan LOBs: An orphan LOB column is a LOB that is found in the LOB table space but that is not referenced by the base table space. If an orphan error is the only type of error reported by CHECK DATA, the base table is considered correct.

An orphan can result from the following situations:

- You recover the base table space to a point in time prior to the insertion of the base table row.
- You recover the base table space to a point in time prior to the definition of the LOB column.
- You recover the LOB table space to a point in time prior to the deletion of a base table row.
- A base record ROWID is incorrect, which results in an orphan LOB column error message and a missing LOB column error message. The missing LOB column error message identifies the ROWID, VERSION and row in error. The missing LOB column is handled depending on the value that you specify for the AUXERROR parameter.

Missing LOBs: A missing LOB column is a LOB that is referenced by the base table space but that is not in the LOB table space. A missing LOB can result from the following situations:

- You recover the LOB table space to a point in time prior to the first insertion of the LOB into the base table.
- You recover the LOB table space to a point in time when the LOB column is null or has a zero length

Out-of-synch LOBs: An out-of-synch LOB error is a LOB that is found in both the base table and the LOB table space, but the LOB in the LOB table space is at a different level. A LOB column is also out-of-synch if the base table is null or has a zero length, but the LOB is found in the LOB table space. An out-of-synch LOB can occur anytime you recover the LOB table space or the base table space to a prior point in time.

Invalid LOBs: An invalid LOB is an uncorrected LOB column error that is found by a previous execution of CHECK DATA AUXERROR INVALIDATE.

Detecting LOB column errors: If you specify either CHECK DATA AUXERROR REPORT or AUXERROR INVALIDATE and a LOB column check error is detected, DB2 issues a message that identifies the table, row, column, and type of error. Any additional actions depend on the option that you specify for the AUXERROR parameter:

- ***When you specify the AUXERROR REPORT option,*** DB2 sets the base table space to the auxiliary CHECK-pending (ACHKP) status. If CHECK DATA encounters only invalid LOB columns and no other LOB column errors, the base table space is set to the auxiliary warning (AUXW) status.
- ***When you specify the AUXERROR INVALIDATE option,*** DB2 sets the base table LOB columns that are in error to an invalid status. DB2 resets the invalid status of LOB columns that have been corrected. If any invalid LOB columns remain in the base table, DB2 sets the base table space to auxiliary warning (AUXW) status. You can use SQL to update a LOB column that is in the AUXW status; however, any other attempt to access the column results in a -904 SQL return code.

See Appendix C, “Advisory or restrictive states,” on page 853 for information about resetting the restrictive table space status.

Resetting auxiliary CHECK-pending status

If a table space has tables with LOB columns and the table space is recovered to a point in time, RECOVER TABLESPACE sets the auxiliary CHECK-pending (ACHKP) status on the table space.

Use one of the following actions to remove the auxiliary CHECK-pending status if DB2 does not find any inconsistencies:

- Use the SCOPE(ALL) option to check all dependent tables in the specified table space. The checks include referential integrity constraints, table check constraints, and the existence of LOB columns.
- Use the SCOPE(PENDING) option to check table spaces or partitions with CHKP status. The checks include referential integrity constraints, table check constraints, and the existence of LOB columns.
- Use the SCOPE(AUXONLY) option to check for LOB columns.

If you specified the AUXERROR(INVALIDATE) option and DB2 finds inconsistencies, it places the table space in AUXW status. See Appendix C, “Advisory or restrictive states,” on page 853 for information about resetting the restrictive table space status.

Terminating or restarting CHECK DATA

This section explains how to terminate and restart the CHECK DATA utility.

Terminating CHECK DATA

When you terminate CHECK DATA, table spaces remain in the same CHECK-pending status as they were at the time the utility was terminated. The CHECKDAT phase places the table space in the CHECK-pending status when CHECK DATA detects an error; at the end of the phase, CHECK DATA resets the CHECK-pending status if it detects no errors. The REPORTCK phase resets the CHECK-pending status if you specify the DELETE YES option.

For instructions on terminating an online utility, see “Terminating an online utility with the TERM UTILITY command” on page 42.

Restarting CHECK DATA

You can restart a CHECK DATA utility job, but it starts from the beginning again. For guidance in restarting online utilities, see "Restarting an online utility" on page 43.

Concurrency and compatibility for CHECK DATA

DB2 treats individual data and index partitions as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible.

Claims and drains: Table 8 shows which claim classes CHECK DATA claims and drains and any restrictive status that the utility sets on the target object. The legend for these claim classes is located at the bottom of the table.

Table 8. Claim classes of CHECK DATA operations

Target objects	CHECK DATA DELETE NO	CHECK DATA DELETE YES	CHECK DATA PART DELETE NO	CHECK DATA PART DELETE YES
Table space or partition	DW/UTRO	DA/UTUT	DW/UTRO	DA/UTUT
Partitioning index or index partition	DW/UTRO	DA/UTUT	DW/UTRO	DA/UTUT
Secondary index	DW/UTRO	DA/UTUT	none	DR
Logical partition of index	none	none	DW/UTRO	DA/UTUT
Primary index	DW/UTRO	DW/UTRO	DW/UTRO	DW/UTRO
RI dependent and descendent table spaces and indexes	none	DA/UTUT	none	DA/UTUT
RI exception table spaces and indexes (FOR EXCEPTION only)	DA/UTUT	DA/UTUT	DA/UTUT	DA/UTUT

Legend:
- DA: Drain all claim classes, no concurrent SQL access
- DR: Drain the repeatable read class, no concurrent access for SQL repeatable readers
- DW: Drain the write claim class, concurrent access for SQL readers
- UTUT: Utility restrictive state, exclusive control
- UTRO: Utility restrictive state, read-only access allowed
- none: Object not affected by this utility
- RI: Referential Integrity

Table 9 shows claim classes on a LOB table space and an index on the auxiliary table.

Table 9. Claim classes of CHECK DATA operations on a LOB table space and index on the auxiliary table

Target objects	CHECK DATA DELETE NO	CHECK DATA DELETE YES
LOB table space	DW/UTRO	DA/UTUT
Index on the auxiliary table	DW/UTRO	DA/UTUT

Table 9. Claim classes of CHECK DATA operations on a LOB table space and index on the auxiliary table (continued)

Target objects	CHECK DATA DELETE NO	CHECK DATA DELETE YES

Legend:
- DW: Drain the write claim class, concurrent access for SQL readers
- DA: Drain all claim classes, no concurrent SQL access
- UTRO: Utility restrictive state, read-only access allowed
- UTUT: Utility restrictive state, exclusive control

When you specify CHECK DATA AUXERROR INVALIDATE, a drain-all is performed on the base table space, and the base table space is set UTUT.

Compatibility: The following utilities are compatible with CHECK DATA and can run concurrently on the same target object:
- DIAGNOSE
- MERGECOPY
- MODIFY
- REPORT
- STOSPACE
- UNLOAD (when CHECK DATA DELETE NO)

SQL operations and other online utilities are incompatible.

To run on DSNDB01.SYSUTILX, CHECK DATA must be the only utility in the job step and the only utility that is running in the DB2 subsystem.

The index on the auxiliary table for each LOB column inherits the same compatibility and concurrency attributes of a primary index.

Sample CHECK DATA control statements

Example 1: Copying violations into exception tables. The control statement in Figure 10 specifies that the CHECK DATA utility is to check for and delete any rows that violate referential and table check constraints in table spaces DSN8D81A.DSN8S81D and DSN8D81A.DSN8S81E. CHECK DATA copies any rows that violate these constraints into the exception tables that are specified in the FOR EXCEPTION clause. For example, CHECK DATA is to copy the violations in table DSN8810.DEPT into table DSN8810.EDEPT.

```
//STEP1    EXEC DSNUPROC,UID='IUIQU1UQ.CHK1',
//         UTPROC='',
//         SYSTEM='DSN'
//SYSUT1   DD DSN=IUIQU1UQ.CHK3.STEP1.SYSUT1,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(8000,(200,20),,,ROUND)
//SYSERR   DD DSN=IUIQU1UQ.CHK3.SYSERR,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(6000,(20,20),,,ROUND)
//SORTOUT  DD DSN=IUIQU1UQ.CHK3.STEP1.SORTOUT,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(6000,(20,20),,,ROUND)
//SYSIN    DD *
CHECK DATA TABLESPACE DSN8D81A.DSN8S81D
           TABLESPACE DSN8D81A.DSN8S81E
     FOR EXCEPTION IN DSN8810.DEPT        USE DSN8810.EDEPT
                   IN DSN8810.EMP         USE DSN8810.EEMP
                   IN DSN8810.PROJ        USE DSN8810.EPROJ
                   IN DSN8810.PROJACT     USE DSN8810.EPROJACT
                   IN DSN8810.EMPPROJACT  USE DSN8810.EEPA
     DELETE YES
//*
```

Figure 10. Example of using the CHECK DATA utility to copy invalid data into exception tables and to delete the invalid data from the original table.

You can create exception tables by using the LIKE clause in the CREATE TABLE statement. For an example of creating an exception table, see "Example: creating an exception table for the project activity table" on page 68.

Example 2: Running CHECK DATA on a table space with LOBs. Before you run CHECK DATA on a table space that contains at least one LOB column, complete the steps that are listed in "For a table with LOB columns" on page 66.

Assume that table space DBIQUQ01.TPIQU01 contains LOB columns. In Figure 11, the control statement with the SCOPE ALL option indicates that CHECK DATA is to check all rows in all dependent tables in table space DBIQUQ01.TPIQU01 for the following violations:

- Violations of referential constraints
- Violations of table check constraints
- Inconsistencies between the base table space and the corresponding LOB table space.

The AUXERROR INVALIDATE option indicates that if the CHECK DATA utility finds a LOB column error in this table space, it is to perform the following actions:

- Issues a warning message
- Sets the base table LOB column to an invalid status
- Sets the base table to auxiliary warning (AUXW) status

```
//STEP11   EXEC DSNUPROC,UID='IUIQU1UQ.CHK2',
//         UTPROC='',
//         SYSTEM='SSTR'
//SYSUT1   DD DSN=IUIQU1UQ.CHK2.STEP5.SYSUT1,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT  DD DSN=IUIQU1UQ.CHK2.STEP5.SORTOUT,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSERR   DD DSN=IUIQU1UQ.CHK2.SYSERR,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
   CHECK DATA TABLESPACE DBIQUQ01.TPIQUQ01 SCOPE ALL
         AUXERROR INVALIDATE
/*
```

Figure 11. Example of running CHECK DATA on a table space with LOBs

Example 3: Specifying the maximum number of exceptions. The control statement in Figure 12 specifies that the CHECK DATA utility is to check all rows in partition number 254 in table space DBNC0216.TPNC0216. The EXCEPTIONS 1 option indicates that the utility is to terminate when it finds one exception. Any exceptions are to be reported by messages only.

```
//CKDATA   EXEC DSNUPROC,UID='L450TST3.CHECK',
//         UTPROC='',
//         SYSTEM='SSTR'
//SYSREC   DD DSN=L450TST3.CHECK.STEP1.SYSREC,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSERR   DD DSN=L450TST3.CHECK.STEP1.SYSERR,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(2000,(20,20),,,ROUND)
//SYSUT1   DD DSN=L450TST3.CHECK.STEP1.SYSUT1,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT  DD DSN=L450TST3.CHECK.STEP1.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
    CHECK DATA TABLESPACE DBNC0216.TPNC0216 PART 254
              SCOPE ALL EXCEPTIONS 1
/*
```

Figure 12. Example of specifying the maximum number of exceptions

Chapter 9. CHECK INDEX

The CHECK INDEX online utility tests whether indexes are consistent with the data that they index, and issues warning messages when it finds an inconsistency.

Run the CHECK INDEX utility after a conditional restart or a point-in-time recovery on all table spaces whose indexes might not be consistent with the data.

Also run CHECK INDEX before running CHECK DATA, especially if you specify DELETE YES. Running CHECK INDEX before CHECK DATA ensures that the indexes that CHECK DATA uses are valid. When checking an auxiliary table index, CHECK INDEX verifies that each LOB is represented by an index entry, and that an index entry exists for every LOB. For more information about running the CHECK DATA utility on a table space that contains at least one LOB column, see "For a table with LOB columns" on page 66.

For a diagram of CHECK INDEX syntax and a description of available options, see "Syntax and options of the CHECK INDEX control statement" on page 80. For detailed guidance on running this utility, see "Instructions for running CHECK INDEX" on page 83.

Output: CHECK INDEX generates several messages that show whether the indexes are consistent with the data. See Part 2 of *DB2 Messages* for more information about these messages.

For unique indexes, any two null values are treated as equal values, unless the index was created with the UNIQUE WHERE NOT NULL clause. In that case, if the key is a single column, it can contain any number of null values, and CHECK INDEX does not issue an error message.

CHECK INDEX issues an error message if it finds two or more null values and the unique index was not created with the UNIQUE WHERE NOT NULL clause.

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:
- STATS privilege for the database
- DBADM, DBCTRL, or DBMAINT authority for the database
- SYSCTRL or SYSADM authority.

An ID with installation SYSOPR authority can also execute CHECK INDEX, but only on a table space in the DSNDB01 or DSNDB06 databases.

If you are using SHRLEVEL CHANGE, the batch user ID that invokes COPY with
the CONCURRENT option must provide the necessary authority to execute the
DFDSS ADRSSU command. DFDSS will create a shadow data set with the
authority of the utility batch address space. The submitter should have a RACF
ALTER authority, or its equivalent, for the shadow data set.

Execution phases of CHECK INDEX:

Phase	Description
UTILINIT	Performs initialization

UNLOAD	Unloads index entries
SORT	Sorts unloaded index entries
CHECKIDX	Scans data to validate index entries
UTILTERM	Performs cleanup

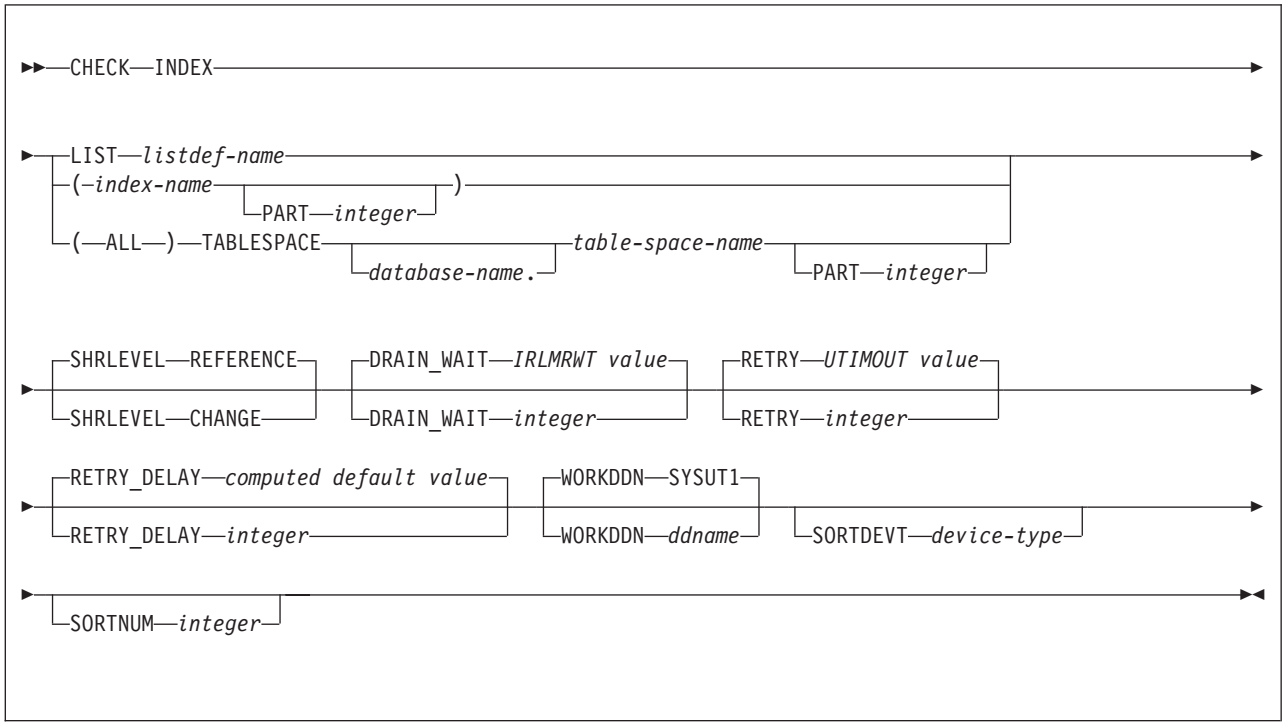
The following topics provide additional information:

- "Syntax and options of the CHECK INDEX control statement"
- "Instructions for running CHECK INDEX" on page 83
- "Concurrency and compatibility for CHECK INDEX" on page 92
- "Sample CHECK INDEX control statements" on page 93

Syntax and options of the CHECK INDEX control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

INDEX Indicates that you are checking for index consistency.

LIST *listdef-name*
Specifies the name of a previously defined LISTDEF list name. The

list should contain only index spaces. Do not specify the name of an index or of a table space. DB2 groups indexes by their related table space and executes CHECK INDEX once per table space. CHECK INDEX allows one LIST keyword for each control statement in CHECK INDEX. For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

(*index-name***, ...)**

Specifies the indexes that are to be checked. All indexes must belong to tables in the same table space. If you omit this option, you must use the (ALL) TABLESPACE option. Then CHECK INDEX checks all indexes on all tables in the table space that you specify.

index-name is the name of an index, in the form *creator-id.name*. If you omit the qualifier *creator-id.*, the user identifier for the utility job is used. If you use a list of names, separate items in the list by commas. Parentheses are required around a name or list of names. Enclose the index name in quotation marks if the name contains a blank.

PART *integer* Identifies a physical partition of a partitioned index or a logical partition of a nonpartitioned index that is to be checked for consistency. If you specify an index on a nonpartitioned table space, an error occurs.

integer is the number of the partition and must be in the range from 1 to the number of partitions that are defined for the table space. The maximum is 4096.

If the PART keyword is not specified, CHECK INDEX tests the entire target index for consistency.

(ALL) Specifies that all indexes in the specified table space that are referenced by the table space are to be checked.

TABLESPACE *database-name.table-space-name*

Specifies the table space from which all indexes are to be checked. If an explicit list of index names is not specified, all indexes on all tables in the specified table space are checked.

Do not specify TABLESPACE with an explicit list of index names.

database-name is the name of the database that the table space belongs to. The **default** is **DSNDB04**.

table-space-name is the name of the table space from which all indexes are checked.

SHRLEVEL Indicates the type of access that is to be allowed for the index, table space, or partition that is to be checked during CHECK INDEX processing.

REFERENCE

Specifies that applications can read from but cannot write to the index, table space, or partition that is to be checked. The **default** is **REFERENCE**.

If you specify SHRLEVEL REFERENCE or use this value as the default, DB2 unloads the index entries, sorts the index entries, and scans the data to validate the index entries.

CHANGE
Specifies that applications can read from and write to the index, table space, or partition that is to be checked.

If you specify SHRLEVEL CHANGE, DB2 performs the following actions:

- Drains all writers and forces the buffers to disk for the specified object and all of its indexes
- Invokes DFSMSdss™ to copy the specified object and all of its indexes to shadow data sets
- Enables read-write access for the specified object and all of its indexes
- Runs CHECK INDEX on the shadow data sets

Note: DFSMSdss uses FlashCopy Version 2 if available. Otherwise, DFSMSdss might take a long time to copy the object, and the time during which the data and indexes have read-only access might increase.

DRAIN_WAIT *integer*
Specifies the number of seconds that CHECK INDEX is to wait when draining the table space or index. The specified time is the aggregate time for objects that are to be checked. This value overrides the values that are specified by the IRLMRWT and UTIMOUT subsystem parameters.

integer can be any integer from 0 to 1800. If you do not specify DRAIN_WAIT or specify a value of 0, CHECK INDEX uses the value of the lock timeout subsystem parameter IRLMRWT.

RETRY *integer* Specifies the maximum number of retries that CHECK INDEX is to attempt.

integer can be any integer from 0 to 255. Specifying a value other than 0 can increase processing costs and result in multiple or extended periods during which the specified index, table space, or partition is in read-only access.

If you do not specify RETRY, CHECK INDEX uses the value of the utility multiplier subsystem parameter UTIMOUT.

RETRY_DELAY *integer*
Specifies the minimum duration, in seconds, between retries. *integer* can be any integer from 1 to 1800.

If you do not specify RETRY_DELAY, CHECK INDEX uses the smaller of the following two values:

- DRAIN_WAIT value × RETRY value
- DRAIN_WAIT value × 10

WORKDDN *ddname*
Specifies a DD statement for a temporary work file.

You can use the WORKDDN keyword to specify either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE

name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

ddname is the DD name. The **default** is **SYSUT1**.

SORTDEVT *device-type*
Specifies the device type for temporary data sets that are to be dynamically allocated by DFSORT. *device-type* is the device type and can be any device type that is acceptable to the DYNALLOC parameter of the SORT or OPTION control statement for DFSORT.

A TEMPLATE specification does not dynamically allocate sort work data sets. The SORTDEVT keyword controls dynamic allocation of these data sets.

SORTNUM *integer*
Specifies the number of temporary data sets that are to be dynamically allocated by the sort program.

integer is the number of temporary data sets that can range from 2 to 255.

If you omit SORTDEVT, SORTNUM is ignored. If you use SORTDEVT and omit SORTNUM, no value is passed to DFSORT; DFSORT uses its own default.

You need at least two sort work data sets for each sort. The SORTNUM value applies to each sort invocation in the utility. For example, if there are three indexes, SORTKEYS is specified, there are no constraints limiting parallelism, and SORTNUM is specified as 8, then a total of 24 sort work data sets will be allocated for a job.

Each sort work data set consumes both above the line and below the link virtual storage, so if you specify too high a value for SORTNUM, the utility may decrease the degree of parallelism due to virtual storage constraints, and possibly decreasing the degree down to one, meaning no parallelism.

Important: The SORTNUM keyword will not be considered if ZPARM UTSORTAL is set to YES and IGNSORTN is set to YES.

Instructions for running CHECK INDEX

To run CHECK INDEX, you must:

1. Prepare the necessary data sets, as described in "Data sets that CHECK INDEX uses" on page 84.
2. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for CHECK INDEX, see "Sample CHECK INDEX control statements" on page 93.)
3. Prepare a utility control statement that specifies the options for the tasks that you want to perform.
4. Check the compatibility table in "Concurrency and compatibility for CHECK INDEX" on page 92 if you want to run other jobs concurrently on the same target objects.
5. Plan for restart if the CHECK INDEX job doesn't complete, as described in "Terminating or restarting CHECK INDEX" on page 91.

6. Run CHECK INDEX by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Data sets that CHECK INDEX uses

Table 10 lists the data sets that CHECK INDEX uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 10. Data sets that CHECK INDEX uses

Data set	Description	Required?
SYSIN	An input data set that contains the utility control statement.	Yes
SYSPRINT	An output data set for messages.	Yes
Work data set	A temporary data set for collecting index key values that are to be checked. Specify its DD name by using the WORKDDN option in the utility control statement. The default DD name is SYSUT1. To find the approximate size in bytes of the work data sets, seeDefining the work data set for CHECK INDEX.	Yes
UTPRINT	A data set that contains messages from DFSORT (usually, SYSOUT or DUMMY).	No

The following object is named in the utility control statement and does not require a DD statement in the JCL:

Index space
Object that is to be checked. (If you want to check only one partition of an index, use the PART option in the control statement.)

DB2 utilities uses DFSORT to perform sorts. Sort work data sets cannot span
volumes. Smaller volumes require more sort work data sets to sort the same
amount of data; therefore, large volume sizes can reduce the number of needed
sort work data sets. It is recommended that at least 1.2 times the amount of data to
be sorted be provided in sort work data sets on disk. For more information about
DFSORT, see *DFSORT Application Programming Guide*.

Defining the work data set for CHECK INDEX

A single sequential data set, which is described by the DD statement in the WORKDDN option, is required during execution of CHECK INDEX.

To find the approximate size in bytes of the WORKDDN data set:

1. For each table, multiply the number of records in the table by the number of indexes on the table that need to be checked.
2. Add the products that you obtain in step 1.
3. Add 8 to the length of the longest key. For nonpadded indexes, the length of the longest key is the maximum possible length of the key with all varying-length columns in the key padded to their maximum length, plus 2 bytes for each varying-length column.
4. Multiply the sum from step 2 by the sum from step 3.

Another method of estimating the size of the WORKDDN data set is to obtain the high-used relative byte address (RBA) for each index from a VSAM catalog listing. Then add the RBAs.

Shadow data sets

When you execute the CHECK INDEX utility with the SHRLEVEL CHANGE option, the utility uses shadow data sets. For user-managed data sets, you must preallocate the shadow data sets before you execute CHECK INDEX SHRLEVEL CHANGE. If a table space, partition, or index resides in DB2-managed data sets and shadow data sets do not already exist when you execute CHECK INDEX, DB2 creates the shadow data sets. At the end of CHECK INDEX processing, the DB2-managed shadow data sets are deleted.

Shadow data set names: Each shadow data set must have the following name:

catname.DSNDB*x*.*dbname*.*psname*.*y*0001.*Lnnn*

In the preceding name, the variables have the following meanings:

variable	meaning
catname	The VSAM catalog name or alias
x	C or D
dbname	Database name
psname	Table space name or index name
y	I or J
Lnnn	Partition identifier. Use one of the following values: • A001 through A999 for partitions 1 through 999 • B000 through B999 for partitions 1000 through 1999 • C000 through C999 for partitions 2000 through 2999 • D000 through D999 for partitions 3000 through 3999 • E000 through E996 for partitions 4000 through 4096

To determine the names of existing shadow data sets, execute one of the following queries against the SYSTABLEPART or SYSINDEXPART catalog tables:

```
SELECT DBNAME, TSNAME, IPREFIX
  FROM SYSIBM.SYSTABLEPART
  WHERE DBNAME = 'dbname' AND TSNAME = 'psname';

SELECT DBNAME, IXNAME, IPREFIX
  FROM SYSIBM.SYSINDEXES X, SYSIBM.SYSINDEXPART Y
  WHERE X.NAME = Y.IXNAME AND X.CREATOR = Y.IXCREATOR
  AND X.DBNAME = 'dbname' AND X.INDEXSPACE = 'psname';
```

For a partitioned table space, DB2 returns rows from which you select the row for the partitions that you want to check.

Defining shadow data sets: Consider the following actions when you preallocate the data sets:

- Allocate the shadow data sets according to the rules for user-managed data sets.
- Define the shadow data sets as LINEAR.
- Use SHAREOPTIONS(3,3).
- Define the shadow data sets as EA-enabled if the original table space or index space is EA-enabled.

- Allocate the shadow data sets on the volumes that are defined in the storage group for the original table space or index space.

If you specify a secondary space quantity, DB2 does not use it. Instead, DB2 uses the SECQTY value for the table space or index space.

Recommendation: Use the MODEL option, which causes the new shadow data set to be created like the original data set. This method is shown in the following example:

```
DEFINE CLUSTER +
   (NAME('catname.DSNDBC.dbname.psname.x0001.L001') +
   MODEL('catname.DSNDBC.dbname.psname.y0001.L001')) +
   DATA                                              +
   (NAME('catname.DSNDBD.dbname.psname.x0001.L001') +
   MODEL('catname.DSNDBD.dbname.psname.y0001.L001') )
```

DB2 treats preallocated shadow data sets as DB2-managed data sets.

Creating shadow data sets for indexes: When you preallocate shadow data sets for indexes, create the data sets as follows:

- Create shadow data sets for the partition of the table space and the corresponding partition in each partitioning index and data-partitioned secondary index.
- Create a shadow data set for logical partitions of nonpartitioned secondary indexes.

Use the same naming scheme for these index data sets as you use for other data sets that are associated with the base index, except use J0001 instead of I0001. For more information about this naming scheme, see the information about the shadow data set naming convention at the beginning of this section, "Shadow data sets" on page 85.

Estimating the size of shadow data sets: If you have not changed the value of FREEPAGE or PCTFREE, the amount of required space for a shadow data set is comparable to the amount of required space for the original data set.

Creating the control statement

Create the utility control statement for the CHECK INDEX job. See "Syntax diagram" on page 80 for CHECK INDEX syntax and option descriptions. See "Sample CHECK INDEX control statements" on page 93 for examples of CHECK INDEX usage.

Instructions for specific tasks

To perform the following tasks, specify the options and values that are documented with your utility control statement.

Checking a single logical partition

You can run CHECK INDEX on a single logical partition of a secondary index. However, what CHECK INDEX can detect is limited as follows:

- CHECK INDEX does not detect duplicate unique keys in different logical partitions. For example, logical partition 1 might have the following keys:

  ```
  A  B  E  F  T  Z
  ```

  Logical partition 2 might have the following keys:

  ```
  M  N  Q  T  V  X
  ```

In this example, the keys are unique within each logical partition, but both logical partitions contain the key, T; so for the index as a whole, the keys are not unique. CHECK INDEX does not detect the duplicates.

- CHECK INDEX does not detect keys that are out of sequence between different logical partitions. For example, the following keys are out of sequence:

```
1  7  5  8  9  10  12
```

If keys 1, 5, 9, and 12 belong to logical partition 1 and keys 7, 8, and 10 belong to logical partition 2, the keys within each partition are in sequence, but the keys for the index, as a whole, are out of sequence, as shown in the following example:

```
LP 1   1     5     9      12
LP 2   7     8     10
```

When checking a single logical partition, CHECK INDEX does not detect this out-of-sequence condition.

Checking indexes in parallel

If you specify more than one index to check and the SHRLEVEL CHANGE option, CHECK INDEX checks the indexes in parallel unless constrained by available memory or sort work files. Checking indexes in parallel reduces the elapsed time for a CHECK INDEX job by sorting the index keys and checking multiple indexes in parallel, rather than sequentially.

Figure 13 shows the flow of a CHECK INDEX job with a parallel index check for a nonpartitioned table space or a single partition of a partitioned table space.

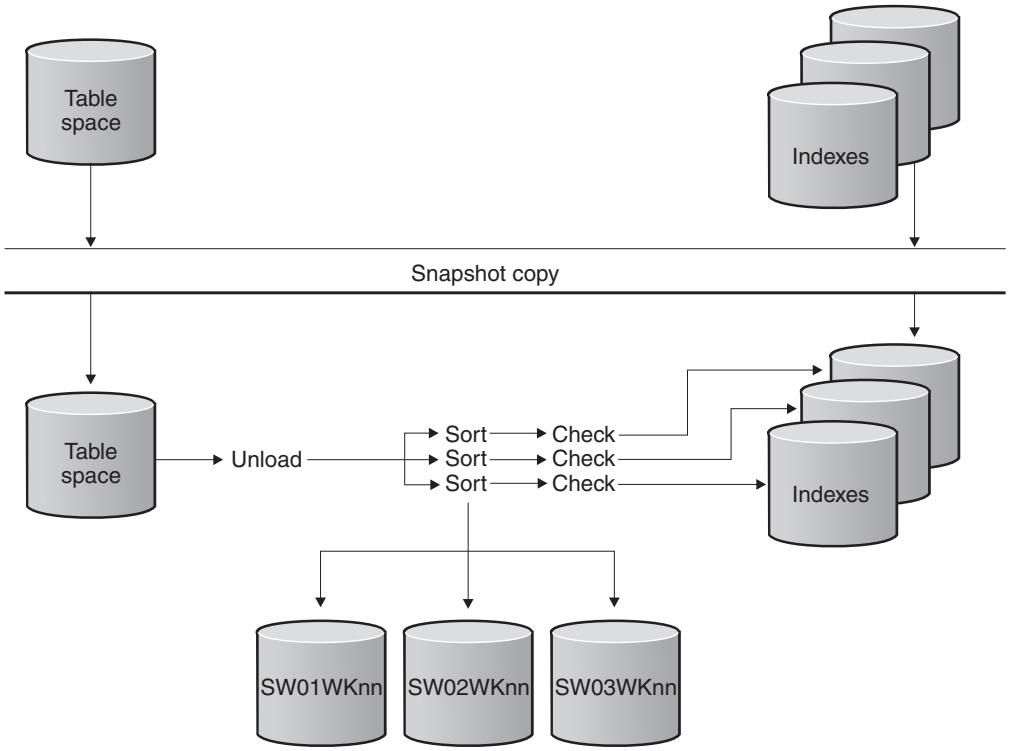


Figure 13. Parallel index check for a nonpartitioned table space or a single partition of a partitioned table space

Figure 14 shows the flow of a CHECK INDEX job with a parallel index check for all partitioning indexes on a partitioned table space.

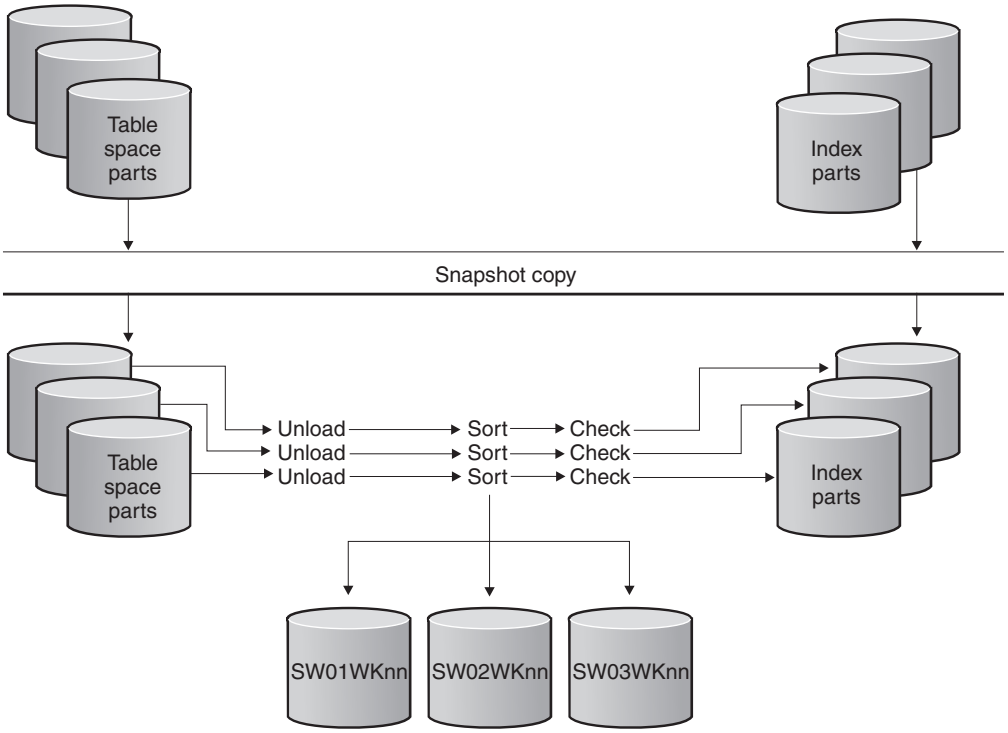


Figure 14. Parallel index check for all partitioning indexes on a partitioned table space

Figure 15 shows the flow of a CHECK INDEX job with a parallel index check for a partitioned table space with a single nonpartitioned secondary index.

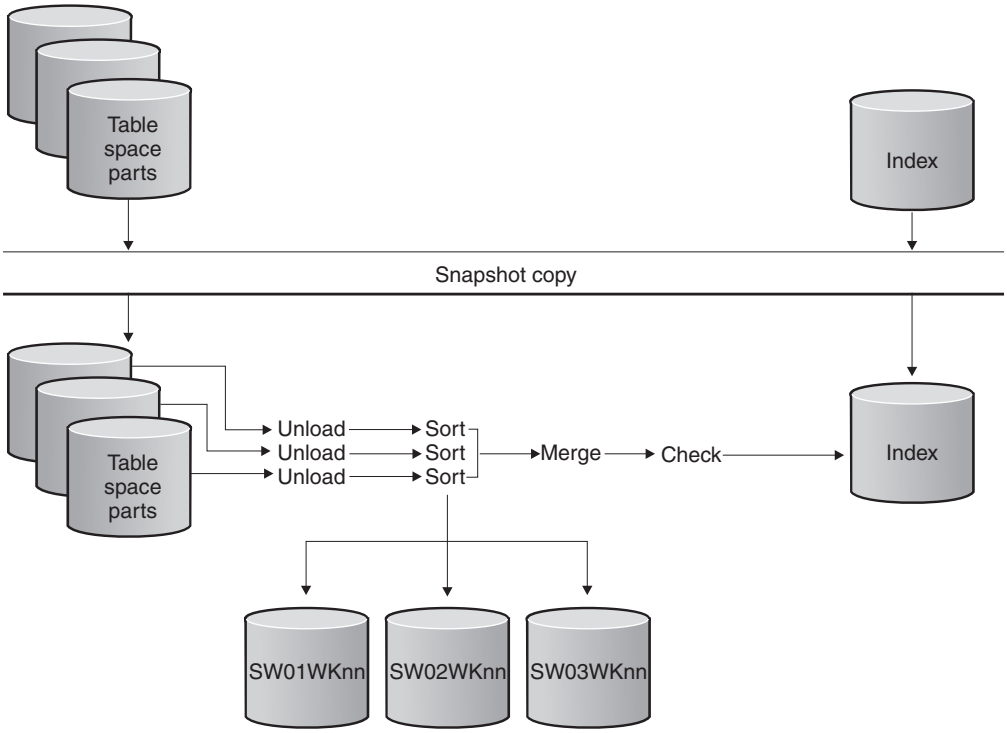


Figure 15. Parallel index check for a partitioned table space with a single nonpartitioned secondary index

Figure 16 shows the flow of a CHECK INDEX job with a parallel index check for all indexes on a partitioned table space.

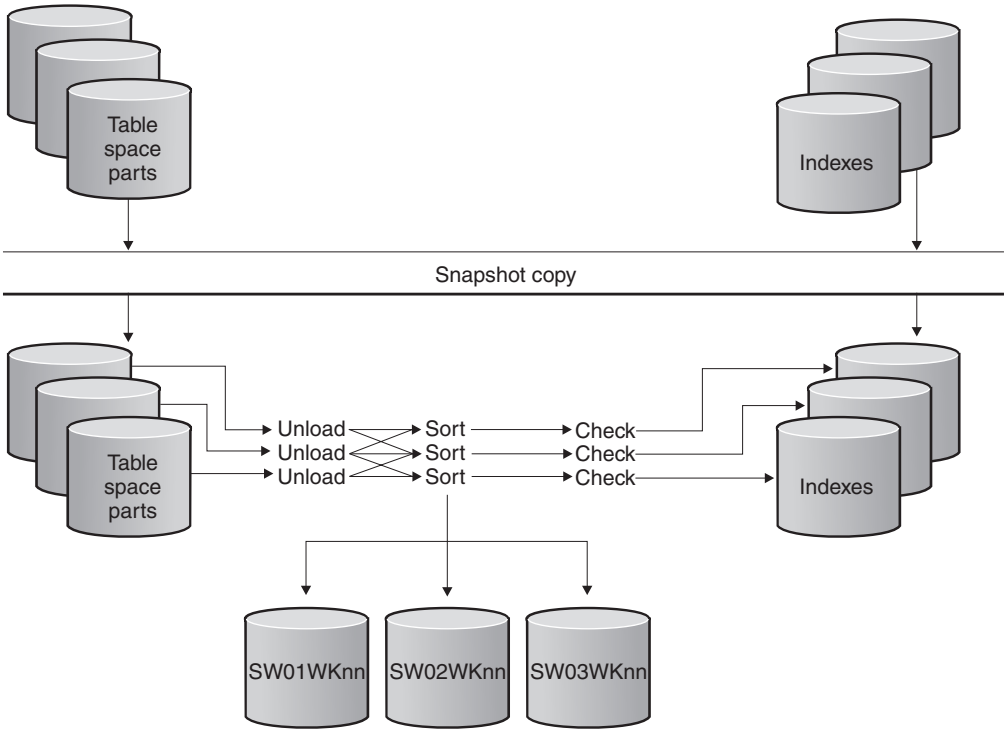


Figure 16. Parallel index check for all indexes on a partitioned table space

Reviewing CHECK INDEX output

CHECK INDEX indicates whether a table space and its indexes are inconsistent, but it does not correct any such inconsistencies. If CHECK INDEX detects inconsistencies, you should analyze the output to determine the problem and then correct the inconsistency. Perform the following actions to identify the inconsistency:

1. Examine the error messages that CHECK INDEX issues.
2. Verify the point in time (TOLOGPOINT, TORBA, or TOCOPY) for each object that is recovered. Use output from REPORT RECOVERY to determine a consistent point for both the table space and its indexes.
3. If the table space is correct, run the REBUILD INDEX utility to rebuild the indexes.
4. If the index is correct, determine a consistent point in time for the table space, and run the RECOVER utility on the table space. Run CHECK INDEX again to verify consistency.
5. If neither the table space nor its indexes are correct, determine a consistent point in time, and then run the RECOVER utility job again, including the table space and its indexes all in the same list.

Terminating or restarting CHECK INDEX

You can terminate CHECK INDEX in any phase without any integrity exposure. For instructions on terminating a utility job, see “Terminating an online utility with the TERM UTILITY command” on page 42.

You can restart a CHECK INDEX utility job, but it starts from the beginning again. For guidance in restarting online utilities, see "Restarting an online utility" on page 43.

Concurrency and compatibility for CHECK INDEX

DB2 treats individual data and index partitions as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible.

Claims and drains: Table 11 shows which claim classes CHECK INDEX claims and drains and any restrictive state that the utility sets on the target object.

Table 11. Claim classes of CHECK INDEX operations

Target	CHECK INDEX SHRLEVEL REFERENCE	CHECK INDEX PART SHRLEVEL REFERENCE	CHECK INDEX SHRLEVEL CHANGE	CHECK INDEX PART SHRLEVEL CHANGE
Table space or partition	DW/UTRO	DW/UTRO	DW/UTRW	DW/UTRW
Partitioning index or index partition	DW/UTRO	DW/UTRO	DW/UTRW	DW/UTRW
Secondary index	DW/UTRO	none	DW/UTRW	DW/UTRW
Data-partitioned secondary index or index partition	DW/UTRO	DW/UTRO	DW/UTRW	DW/UTRW
Logical partition of an index	none	DW/UTRO	DW/UTRW	DW/UTRW

Legend:
- DW: Drain the write claim class, concurrent access for SQL readers
- UTRO: Utility restrictive state, read only-access allowed
- UTRW: Utility restrictive state, read and write access allowed
- none: Object not affected by this utility

CHECK INDEX does not set a utility restrictive state if the target object is DSNDB01.SYSUTILX.

Compatibility: Table 12 shows which utilities can run concurrently with CHECK INDEX on the same target object. The first column lists the other utility and the second column lists whether or not that utility is compatible with CHECK INDEX. The target object can be a table space, an index space, or an index partition. If compatibility depends on particular options of a utility, that information is also documented in the table.

Table 12. Compatibility of CHECK INDEX SHRLEVEL REFERENCE with other utilities

Action	Compatible with CHECK INDEX?
CHECK DATA	No
CHECK INDEX.	Yes
CHECK LOB	Yes
COPY INDEXSPACE	Yes
COPY TABLESPACE	Yes
DIAGNOSE	Yes
LOAD	No

Table 12. Compatibility of CHECK INDEX SHRLEVEL REFERENCE with other utilities (continued)

Action	Compatible with CHECK INDEX?
MERGECOPY	Yes
MODIFY	Yes
QUIESCE	Yes
REBUILD INDEX	No
RECOVER INDEX	No
RECOVER TABLESPACE	No
REORG INDEX	No
REORG TABLESPACE UNLOAD CONTINUE or PAUSE	No
REORG TABLESPACE UNLOAD ONLY or EXTERNAL	Yes
REPAIR DELETE or REPLACE	No
REPAIR DUMP or VERIFY	Yes
REPORT	Yes
RUNSTATS	Yes
STOSPACE	Yes
UNLOAD	Yes

To run on SYSIBM.DSNLUX01 or SYSIBM.DSNLUX02, CHECK INDEX must be the only utility within the job step.

Sample CHECK INDEX control statements

Example 1: Checking all indexes. The control statement in Figure 17 specifies that the CHECK INDEX utility is to check all indexes in sample table space DSN8D81A.DSN8S81E.

```
//STEP1   EXEC DSNUPROC,UID='IUIQU1UQ.CHK1',
//         UTPROC='',
//         SYSTEM='DSN'
//SYSUT1   DD DSN=IUIQU1UQ.CHK3.STEP1.SYSUT1,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(8000,(200,20),,,ROUND)
//SYSERR   DD DSN=IUIQU1UQ.CHK3.SYSERR,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(6000,(20,20),,,ROUND)
//SORTOUT  DD DSN=IUIQU1UQ.CHK3.STEP1.SORTOUT,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(6000,(20,20),,,ROUND)
//SYSIN    DD *
CHECK INDEX (ALL) TABLESPACE DSN8D81A.DSN8S81E
//*
```

Figure 17. Example of checking all indexes

Example 2: Checking one index. The following control statement specifies that the CHECK INDEX utility is to check the project-number index (DSN8810.XPROJ1) on the sample project table. SORTDEVT SYSDA specifies that SYSDA is the device type for temporary data sets that are to be dynamically allocated by DFSORT.

```
CHECK INDEX (DSN8810.XPROJ1)
  SORTDEVT SYSDA
```

Example 3: Checking more than one index. The following control statement specifies that the CHECK INDEX utility is to check the indexes DSN8810.XEMPRAC1 and DSN8810.XEMPRAC2 on the employee-to-project-activity sample table.

```
CHECK INDEX NAME (DSN8810.XEMPRAC1, DSN8810.XEMPRAC2)
```

Example 4: Checking partitions of all indexes. In the following control statement, table space DB0S0301.TP0S0301 has one partitioned index (ADMF001.IP0S0301), one data-partitioned secondary index (ADMF001.ID0S0302), and one nonpartitioned secondary index (ADMF001.IX0S0303). The (ALL) option indicates that all three indexes on the table space are to be checked. PART 3 indicates that CHECK INDEX is to check the third physical partition of any partitioned indexes and the third logical partition of any nonpartitioned indexes.

```
CHECK INDEX(ALL) TABLESPACE DBOS0301.TPOS0301 PART 3 SORTDEVT SYSDA
```

In this case, CHECK INDEX checks the third physical partition of ADMF001.IP0S0301, the third physical partition of ADMF001.ID0S0302, and the third logical partition of ADMF001.IX0S0303, as indicated by the output in Figure 18.

```
DSNU050I  DSNUGUTC- CHECK INDEX(ALL) TABLESPACE DBOS0301.TPOS0301 PART 3 SORTDEVT SYSDA
DSNU700I= DSNUKGET- 10 INDEX ENTRIES UNLOADED FROM INDEX='ADMF001.IPOS0301' PARTITION=3
DSNU700I= DSNUKGET- 10 INDEX ENTRIES UNLOADED FROM INDEX='ADMF001.IDOS0302' PARTITION=3
DSNU701I= DSNUKGET- 10 INDEX ENTRIES UNLOADED FROM 'ADMF001.IXOS0303'
DSNU705I  DSNUK001- UNLOAD PHASE COMPLETE - ELAPSED TIME=00:00:00
DSNU717I= DSNUKTER- 10 ENTRIES CHECKED FOR INDEX 'ADMF001.IPOS0301' PARTITION=3
DSNU717I= DSNUKTER- 10 ENTRIES CHECKED FOR INDEX 'ADMF001.IDOS0302' PARTITION=3
DSNU717I= DSNUKTER- 10 ENTRIES CHECKED FOR INDEX 'ADMF001.IXOS0303' PARTITION=3
DSNU720I  DSNUK001- CHECKIDX PHASE COMPLETE, ELAPSED TIME=00:00:00
DSNU010I  DSNUGBAC- UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 18. CHECK INDEX output from a job that checks the third partition of all indexes.

Example 5: Checking indexes in a list. The LISTDEF control statement in Figure 19 on page 95 defines a list of indexes called CHKIDXB_LIST. For more information about the options for LISTDEF control statements, see Chapter 15, “LISTDEF,” on page 173. The CHECK INDEX control statement specifies that CHECK INDEX is to check all indexes that are included in the CHKIDXB_LIST list. WORKDDN SYSUT1 specifies that SYSUT1 is the DD name of the temporary work file for sort input; SYSUT1 is the default. SORTDEVT SYSDA specifies that SYSDA is the device type for temporary data sets that are to be dynamically allocated by DFSORT. SORTNUM 4 specifies that four of these data sets are to be dynamically allocated.

```
//CHKIDXB EXEC PGM=DSNUTILB,REGION=4096K,PARM='SSTR,CHKINDX1'
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//UTPRINT  DD SYSOUT=A
//DSNTRACE DD SYSOUT=A
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03
//SYSOUT   DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03
//SORTLIB  DD DISP=SHR,DSN=SYS1.SORTLIB
//SORTOUT  DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03
//SORTWK04 DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03
//SYSERR   DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03
//SYSIN    DD *
LISTDEF CHKIDXB_LIST INCLUDE INDEXSPACE DBOT55*.* ALL
CHECK INDEX LIST CHKIDXB_LIST
                 WORKDDN SYSUT1
                 SORTDEVT SYSDA
                 SORTNUM 4
/*
```

Figure 19. Example of checking indexes in a list

Chapter 10. CHECK LOB

You can run the CHECK LOB online utility on a LOB table space to identify any structural defects in the LOB table space and any invalid LOB values.

Run the CHECK LOB utility in the following circumstances:

- Run the utility on a LOB table space that is in CHECK-pending (CHKP) status to identify structural defects. If none are found, the CHECK LOB utility turns the CHKP status off.
- Run the utility on a LOB table space that is in auxiliary-warning (AUXW) status to identify invalid LOBs. If none exist, the CHECK LOB utility turns AUXW status off.
- Run the utility after a conditional restart or a point-in-time recovery on all table spaces where LOB table spaces might not be synchronized.
- Run the utility before you run the CHECK DATA utility on a table space that contains at least one LOB column. For more information about running the CHECK DATA utility on table spaces with LOB columns, see "For a table with LOB columns" on page 66.

For a diagram of CHECK LOB syntax and a description of available options, see "Syntax and options of the CHECK LOB control statement" on page 98. For detailed guidance on running this utility, see "Instructions for running CHECK LOB" on page 99.

Output: After successful execution, CHECK LOB resets the CHECK-pending (CHKP) and auxiliary-warning (AUXW) statuses.

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:

- STATS privilege for the database
- DBADM, DBCTRL, or DBMAINT authority for the database
- SYSCTRL or SYSADM authority

An ID with installation SYSOPR authority can also execute CHECK LOB.

Execution phases of CHECK LOB:

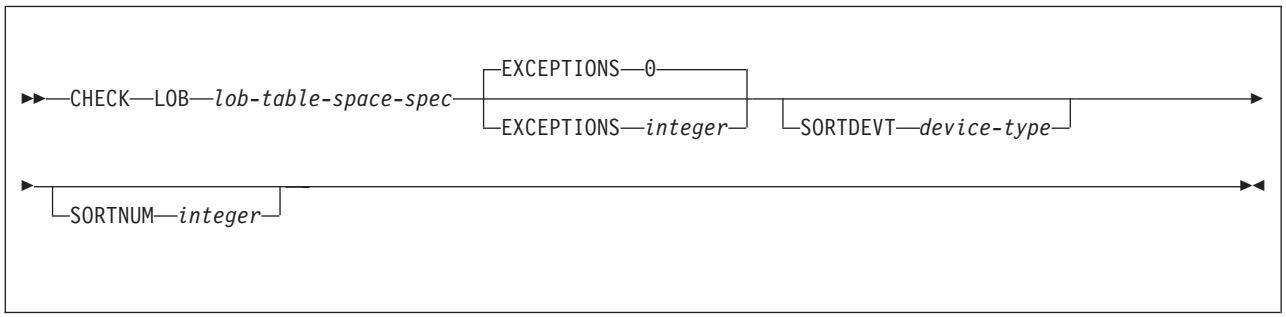
Phase	Description
UTILINIT	Performs initialization
CHECKLOB	Scans all active pages of the LOB table space; generates up to four records per LOB page; passes records to the SORTIN phase
SORTIN	Passes CHECKLOB phase records to SORT
SORT	Sorts the records from the CHECKLOB phase
SORTOUT	Passes sorted records to the REPRTLOB phase
REPRTLOB	Examines records that are produced by the CHECKLOB phase; issues error messages
UTILTERM	Performs cleanup

The following topics provide additional information:

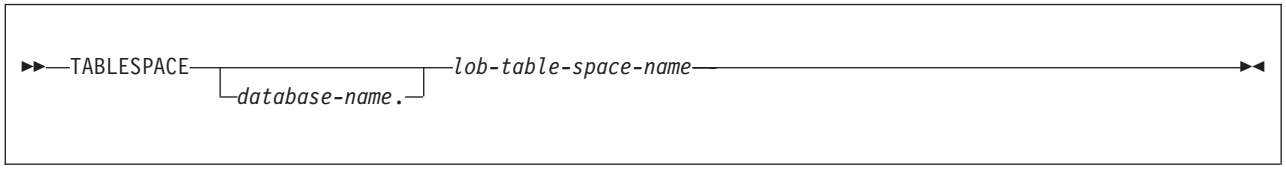
Syntax and options of the CHECK LOB control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram



lob-table-space-spec:



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

LOB Indicates that you are checking a LOB table space for defects.

TABLESPACE *database-name.lob-table-space-name*
Specifies the table space to which the data belongs.

database-name is the name of the database and is optional. The **default** is **DSNDB04**.

lob-table-space-name is the name of the LOB table space.

EXCEPTIONS *integer*
Specifies the maximum number of exceptions, which are reported by messages only. CHECK LOB terminates in the CHECKLOB phase when it reaches the specified number of exceptions.

All defects that are reported by messages are applied to the exception count.

integer is the maximum number of exceptions. The **default** is **0**, which indicates no limit on the number of exceptions.

SORTDEVT *device-type*
Specifies the device type for temporary data sets that are to be dynamically allocated by DFSORT.

A TEMPLATE specification does not dynamically allocate sort work data sets. The SORTDEVT keyword controls dynamic allocation of these data sets.

device-type is the device type and can be any device type that is acceptable to the DYNALLOC parameter of the SORT or OPTION control statement for DFSORT, as described in *DFSORT Application Programming: Guide.*

SORTNUM *integer*
Indicates the number of temporary data sets that are to be dynamically allocated by the sort program.

integer is the number of temporary data sets that can range from 2 to 255.

If you omit SORTDEVT, SORTNUM is ignored. If you use SORTDEVT and omit SORTNUM, no value is passed to DFSORT, which then uses its own default.

You need at least two sort work data sets for each sort. The SORTNUM value applies to each sort invocation in the utility. For example, if there are three indexes, SORTKEYS is specified, there are no constraints limiting parallelism, and SORTNUM is specified as 8, then a total of 24 sort work data sets will be allocated for a job.

Each sort work data set consumes both above the line and below the link virtual storage, so if you specify too high a value for SORTNUM, the utility may decrease the degree of parallelism due to virtual storage constraints, and possibly decreasing the degree down to one, meaning no parallelism.

Important: The SORTNUM keyword will not be considered if ZPARM UTSORTAL is set to YES and IGNSORTN is set to YES.

Instructions for running CHECK LOB

To run CHECK LOB:

1. Read "Before running CHECK LOB" on page 100.
2. Prepare the necessary data sets, as described in "Data sets that CHECK LOB uses" on page 100.
3. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for CHECK LOB, see "Sample CHECK LOB control statements" on page 102.)
4. Prepare a utility control statement that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks" on page 100.
5. Check the compatibility table in "Concurrency and compatibility for CHECK LOB" on page 102 if you want to run other jobs concurrently on the same target objects.

6. Plan for restarting CHECK LOB if the job doesn't complete, as described in "Terminating or restarting CHECK LOB" on page 101.
7. Run CHECK LOB by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Before running CHECK LOB

You must first recover a LOB table space that is in RECOVER-pending status before running CHECK LOB.

Data sets that CHECK LOB uses

Table 13 lists the data sets that CHECK LOB uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 13. Data sets that CHECK LOB uses

Data set	Description	Required?
SYSIN	An input data that contains the utility control statement.	Yes
SYSPRINT	An output data set for messages.	Yes
UTPRINT	A data set that contains messages from DFSORT (usually, SYSOUT or DUMMY).	No

The following object is named in the utility control statement and does not require DD statements in the JCL:

Table space
Object that is to be checked.

DB2 utilities uses DFSORT to perform sorts. Sort work data sets cannot span volumes. Smaller volumes require more sort work data sets to sort the same amount of data; therefore, large volume sizes can reduce the number of needed sort work data sets. It is recommended that at least 1.2 times the amount of data to be sorted be provided in sort work data sets on disk. For more information about DFSORT, see *DFSORT Application Programming Guide*.

Creating the control statement

Create the utility control statement for the CHECK LOB job. See "Syntax diagram" on page 98 for CHECK LOB syntax and option descriptions. See "Sample CHECK LOB control statements" on page 102 for examples of CHECK LOB usage.

Beginning in Version 8, the CHECK LOB utility does not require SYSUT1 and SORTOUT data sets. Work records are written to and processed from an asynchronous SORT phase. The WORKDDN keyword, which provided the DD names of the SYSUT1 and SORTOUT data sets in earlier versions of DB2, is not needed and is ignored. You do not need to modify existing control statements to remove the WORKDDN keyword.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in your utility control statement:

"Finding and resolving violations" on page 101
"Resetting CHECK-pending status for a LOB table space" on page 101

"Resolving media failure"

Finding and resolving violations

CHECK LOB issues message DSNU743I whenever it finds a LOB value that is invalid. The violation is identified by:

- The row ID and version number of the LOB
- A reason code for the error
- The page number where the error was found

You can resolve LOB violations by using the UPDATE or DELETE SQL statements to update the LOB column or delete the row that is associated with the LOB. (Use the row ID from message DSNU743I.) For more information, see UPDATE or DELETE in Chapter 5 of *DB2 SQL Reference*.

If CHECK LOB issues either message DSNU785I or DSNU787I, it has detected a logical inconsistency within the LOB table space. Contact IBM Software Support for assistance with diagnosing and resolving the problem.

Resetting CHECK-pending status for a LOB table space

If you run CHECK LOB and LOB table space errors are found, the table space is placed in CHECK-pending status.

Complete the following tasks to remove the CHECK-pending status:

1. Correct any defects that are found in the LOB table space by using the REPAIR utility.
2. To reset CHECK-pending or auxiliary-warning status, run CHECK LOB again, or run the REPAIR utility.

Use the REPAIR utility with care, as improper use can further damage the data. If necessary, contact IBM Software Support for guidance on using the REPAIR utility.

Resolving media failure

Run CHECK LOB on a LOB table space after experiencing a media failure that leaves LOB pages in the logical page list (LPL).

Terminating or restarting CHECK LOB

This section describes how to terminate and restart the CHECK LOB utility.

Terminating CHECK LOB

If you terminate CHECK LOB during the CHECKLOB phase, LOB table spaces remain in CHECK-pending status. During normal execution, the CHECKLOB phase places the LOB table space in CHECK-pending status; at the end of the phase, the CHECK-pending status is reset if no errors are detected.

For instructions on terminating an online utility, see "Terminating an online utility with the TERM UTILITY command" on page 42.

Restarting CHECK LOB

You can restart a CHECK LOB utility job, but it starts from the beginning again. For guidance in restarting online utilities, see "Restarting an online utility" on page 43.

Concurrency and compatibility for CHECK LOB

DB2 treats individual data and index partitions as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible.

Claims and drains: Table 14 shows which claim classes CHECK LOB claims and drains and any restrictive state that the utility sets on the target object.

Table 14. Claim classes for CHECK LOB operations on a LOB table space and index on the auxiliary table

Target objects	CHECK LOB
LOB table space	DW/UTRO
Index on the auxiliary table	DW/UTRO

Legend:
- DW: Drain the write claim class, concurrent access for SQL readers
- UTRO: Utility restrictive state, read-only access allowed

Compatibility: Any SQL operation or other online utility that attempts to update the same LOB table space is incompatible.

Sample CHECK LOB control statements

Example: Checking a LOB table space. The following control statement specifies that the CHECK LOB utility is to check LOB table space DBIQUG01.TLIQUG02 for structural defects or invalid LOB values. The EXCEPTIONS 3 option indicates that the CHECK LOB utility is to terminate when it finds three exceptions. The SORTDEVT and SORTNUM options provide information about temporary data sets that are to be dynamically allocated by DFSORT. SORTDEVT SYSDA specifies that the device type is SYSDA, and SORTNUM 4 indicates that four temporary data sets are to be dynamically allocated by the sort program.

```
//STEP1    EXEC DSNUPROC,UID='IUIQU2UG.CHECKL',
//         UTPROC='',
//         SYSTEM='DSN'
//SYSIN    DD *
CHECK LOB TABLESPACE DBIQUG01.TLIQUG02
      EXCEPTIONS 3 SORTDEVT SYSDA
      SORTNUM 4
```

Chapter 11. COPY

The COPY online utility creates up to four image copies of any of the following objects:

- Table space
- Table space partition
- Data set of a linear table space
- Index space
- Index space partition

The two types of image copies are:

1. A *full image copy*, which is a copy of all pages in a table space, partition, data set, or index space.
2. An *incremental image copy*, which is a copy only of those data pages that have been modified since the last use of the COPY utility and system pages.

The RECOVER utility uses these copies when recovering a table space or index space to the most recent time or to a previous time. Copies can also be used by the MERGECOPY, RECOVER, COPYTOCOPY, and UNLOAD utilities.

You can copy a list of objects in parallel to improve performance. Specifying a list of objects along with the SHRLEVEL REFERENCE option creates a single recovery point for that list of objects. Specifying the PARALLEL keyword allows you to copy a list of objects in parallel, rather than serially.

To calculate the number of threads you need when you specify the PARALLEL keyword, use the formula (n * 2 + 1), where n is the number of objects that are to be processed in parallel, regardless of the total number of objects in the list. If you do not use the PARALLEL keyword, n is one and COPY uses three threads for a single-object COPY job.

For a diagram of COPY syntax and a description of available options, see "Syntax and options of the COPY control statement" on page 104. For detailed guidance on running this utility, see "Instructions for running COPY" on page 114.

Output: Output from the COPY utility consists of:

- Up to four sequential data sets containing the image copy.
- Rows in the SYSIBM.SYSCOPY catalog table that describe the image copy data sets that are available to the RECOVER utility. Your installation is responsible for ensuring that these data sets are available if the RECOVER utility requests them.
- If you specify the CHANGELIMIT option, a report on the change status of the table space.

The COPY-pending status is set off for table spaces if the copy was a full image copy. However, DB2 does not reset the COPY-pending status if you copy a single piece of a multi-piece linear data set. If you copy a single table space partition, DB2 resets the COPY-pending status only for the copied partition and not for the whole table space. DB2 resets the informational COPY-pending (ICOPY) status after you copy an index space or index.

Related information: See Part 4 (Volume 1) of *DB2 Administration Guide* for uses of COPY in the context of planning for database recovery. For information about

creating inline copies during LOAD, see “Using inline COPY with LOAD” on page 252. You can also create inline copies during REORG; see “Using inline copy with REORG TABLESPACE” on page 470 for more information.

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:
- IMAGCOPY privilege for the database
- DBADM, DBCTRL, or DBMAINT authority for the database
- SYSCTRL or SYSADM authority

An ID with installation SYSOPR authority can also execute COPY, but only on a table space in the DSNDB01 or DSNDB06 database.

The batch user ID that invokes COPY with the option must provide the necessary authority to execute the DFDSS DUMP command.

Execution phases of COPY: The COPY utility operates in these phases:

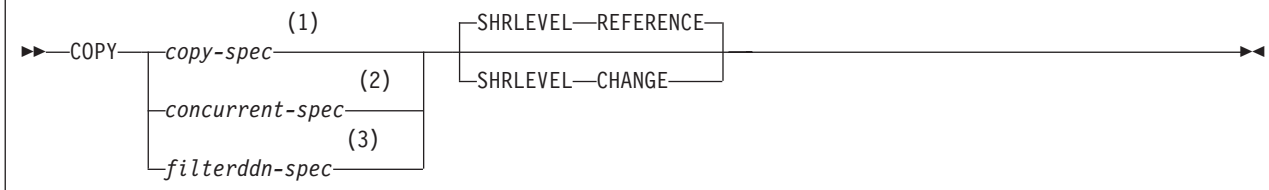
Phase	Description
UTILINIT	Performs initialization and setup
REPORT	Reports for CHANGELIMIT option
COPY	Copies
UTILTERM	Performs cleanup

The following topics provide additional information:
- “Syntax and options of the COPY control statement”
- “Instructions for running COPY” on page 114
- “Concurrency and compatibility for COPY” on page 128
- “Sample COPY control statements” on page 131

Syntax and options of the COPY control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram



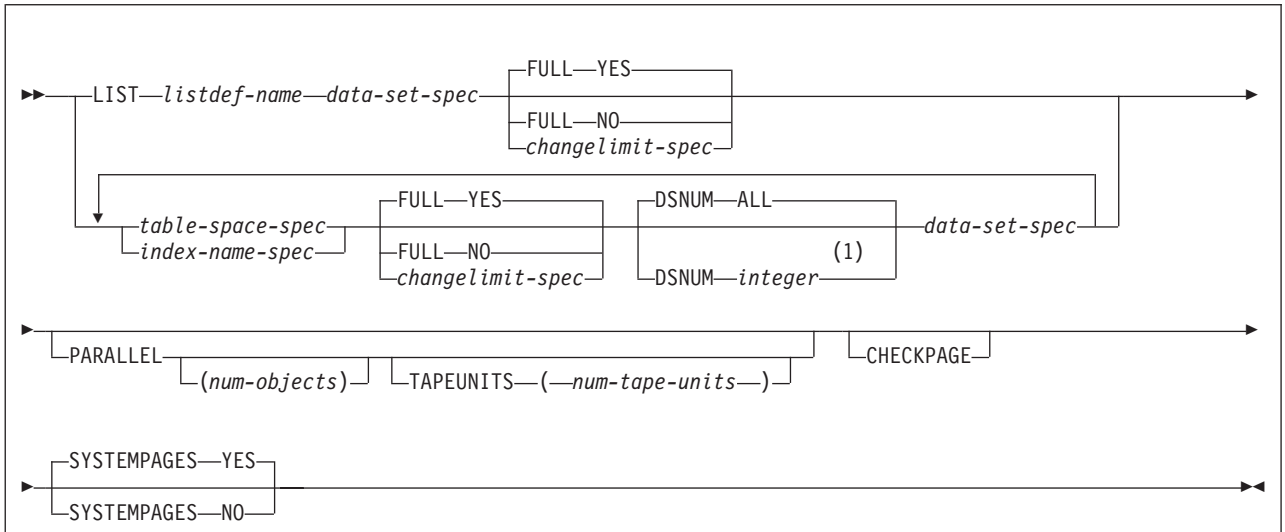
Notes:

1 Use the copy-spec if you do not want to use the CONCURRENT option.

2 Use the concurrent-spec if you want to use the CONCURRENT option, but not the FILTERDDN option.

3 Use the filterddn spec if you want to use the CONCURRENT and FILTERDDN options.

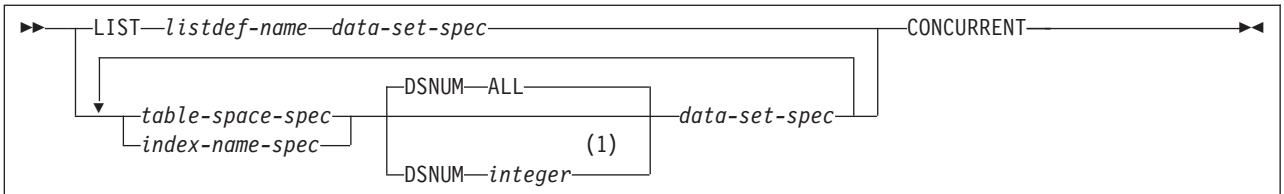
copy-spec:



Notes:

1 Not valid for nonpartioning indexes.

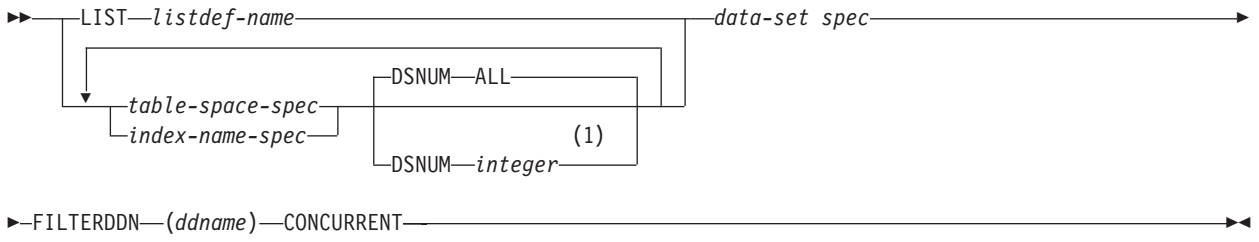
concurrent-spec:



Notes:

1 Not valid for nonpartioning indexes.

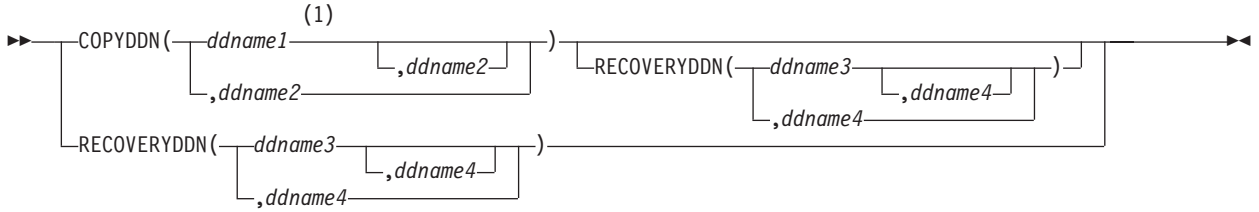
filterddn-spec:



Notes:

1 Not valid for nonpartioning indexes.

data-set-spec:



Notes:

1 COPYDDN SYSCOPY is the default for the primary copy, but this default can only be used for one object in the list.

changelimit-spec:

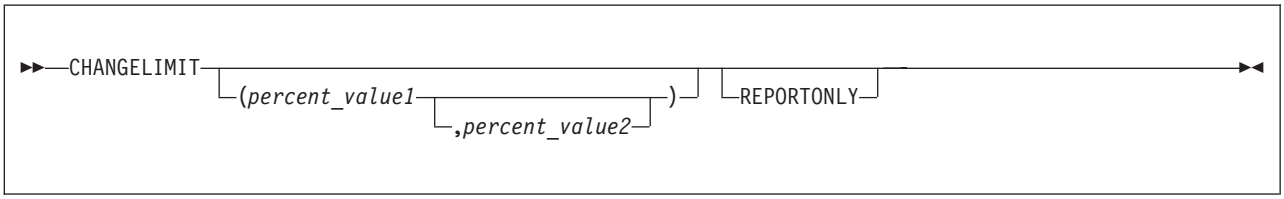
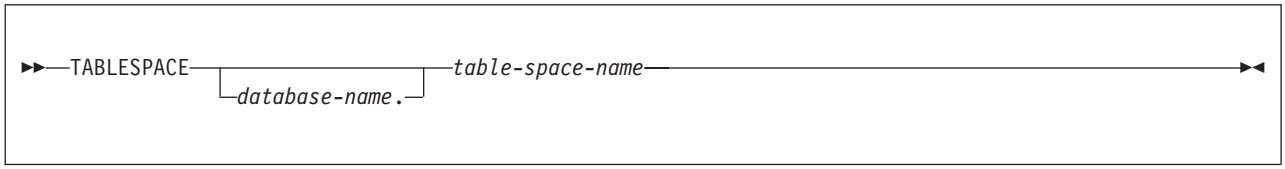
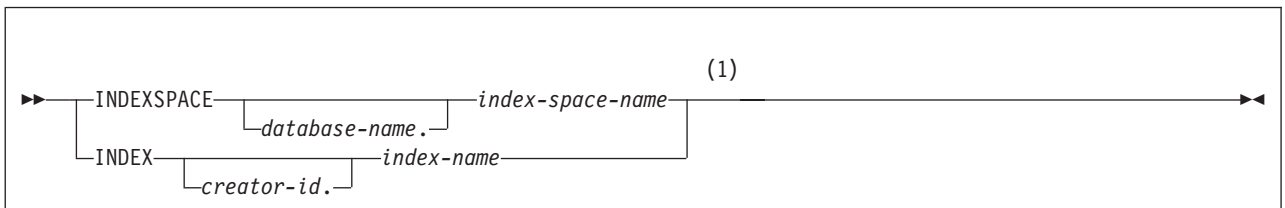


table-space-spec:



index-name-spec:



Notes:

1 INDEXSPACE is the preferred specification.

Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

LIST *listdef-name*

Specifies the name of a previously defined LISTDEF list name. LIST specifies one LIST keyword for each COPY control statement. Do not specify LIST with either the INDEX or the TABLESPACE keyword. DB2 invokes COPY once for the entire list. For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

TABLESPACE*database-name.table-space-name*

Specifies the table space (and, optionally, the database it belongs to) that is to be copied.

database-name is the name of the database that the table space belongs to. The **default** is **DSNDB04**.

table-space-name is the name of the table space to be copied.

Specify the DSNDB01.SYSUTILX, DSNDB06.SYSCOPY, or DSNDB01.SYSLGRNX table space by itself in a single COPY statement. Alternatively, specify the DSNDB01.SYSUTILX, DSNDB06.SYSCOPY, or DSNDB01.SYSLGRNX table space with indexes over the table space that were defined with the COPY YES attribute.

INDEXSPACE *database-name.index-space-name*

Specifies the qualified name of the index space that is to be copied; the name is obtained from the SYSIBM.SYSINDEXES table. The specified index space must be defined with the COPY YES attribute.

database-name Optionally specifies the name of the database that the index space belongs to. The **default** is **DSNDB04**.

index-space-name specifies the name of the index space that is to be copied.

INDEX *creator-id.index-name*
Specifies the index that is to be copied. Enclose the index name in quotation marks if the name contains a blank.

creator-id optionally specifies the creator of the index. The **default** is the user identifier for the utility.

index-name specifies the name of the index that is to be copied.

COPYDDN (*ddname1,ddname2***)**
Specifies a DD name or a TEMPLATE name for the primary (*ddname1*) and backup (*ddname2*) copy data sets for the image copy at the local site.

You can use the **COPYDDN** keyword to specify either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

ddname is the DD name. The **default** is **SYSCOPY** for the primary copy. You can use the default for only one object in the list. The first object in the list that does not have COPYDDN specified uses the default. Any other objects in the list that do not have COPYDDN specified cause an error.

If you use the CHANGELIMIT REPORTONLY option, you can use a DD DUMMY statement when you specify the SYSCOPY output data set. This card prevents a data set from being allocated and opened.

Recommendation: Catalog all of your image copy data sets.

You cannot have duplicate image copy data sets. If the DD statement identifies a noncataloged data set with the same name, volume serial, and file sequence number as one that is already recorded in the SYSIBM.SYSCOPY catalog table, the COPY utility issues a message and does not make an image copy. If COPY identifies a cataloged data set with only the same name, it does not make an image copy. For cataloged image copy data sets, CATLG must be specified for the normal termination disposition in the DD statement, as shown in the following example:

```
DISP=(MOD,CATLG,CATLG)
```

The DSVOLSER field of the SYSCOPY entry is blank.

If you use the CONCURRENT and FILTERDDN options, ensure that the size of the copy data set is large enough to include all of the objects in the list.

RECOVERYDDN (*ddname3,ddname4***)**
Specifies a DD name or a template name for the primary (*ddname3*) and backup (*ddname4*) copy data sets for the image copy at the recovery site.

You can use the **RECOVERYDDN** keyword to specify either a DD name or a template name. If utility processing detects that the

specified name is both a DD name in the current job step and a template name, the utility THE uses the DD name. For more information about template specifications, see Chapter 31, "TEMPLATE," on page 593.

ddname3 and *ddname4* are DD names.

You cannot have duplicate image copy data sets.

If you use the CONCURRENT and FILTERDDN options, ensure that the size of the copy data set is large enough to include all of the objects in the list.

FULL Specifies that COPY is to make either a full or an incremental image copy.

YES Specifies a full image copy. Making a full image copy resets the COPY-pending status for the table space or index, or for the partition if you specify DSNUM. The **default** is **YES**.

NO Specifies only an incremental image copy. Only changes since the last image copy are to be copied. **NO** is not valid for indexes.

Incremental image copies are not allowed in the following situations:

- The last full image copy of the table space was taken with the CONCURRENT option.
- No full image copies exist for the table space or data set that is being copied.
- After a successful LOAD or REORG operation, unless an inline copy was made during the LOAD or REORG job.
- You specify one of the following table spaces: DSNDB01.DBD01, DSNDB01.SYSUTILX, or DSNDB06.SYSCOPY.
- A previous COPY was terminated with the -TERM UTIL command, so the most recent SYSIBM.SYSCOPY record for the object contains ICTYPE = T.

For incremental image copies of partitioned table spaces, COPY includes the header page for each partition that has changed pages.

COPY automatically takes a full image copy of a table space if you specify FULL NO when an incremental image copy is not allowed.

CHANGELIMIT

Specifies the percentage limit of changed pages in the table space, partition, or data set at which an incremental or full image copy is to be taken.

percent_value1

Specifies the first value in the CHANGELIMIT range. *percent_value1* must be an integer or decimal value from 0.0 to 100.0. You do not need to specify leading zeroes, and the decimal point is not required when specifying a whole integer. Specify a maximum of one decimal place for a decimal value. For example, you can specify `.5`. If you specify this value,

COPY takes an incremental image copy if more than one half of one percent of the pages have changed.

percent_value2
Specifies the second value in the CHANGELIMIT range. *percent_value2* must be an integer or decimal value from 0.0 to 100.0. You do not need to specify leading zeroes, and the decimal point is not required when specifying a whole integer. Specify a maximum of one decimal place for a decimal value (for example, `.5`).

COPY CHANGELIMIT accepts values in any order. For example, you can specify (10,1) or (1,10).

If only one value is specified, COPY CHANGELIMIT:

- Creates an incremental image copy if the percentage of changed pages is greater than 0 and less than *percent_value1*.
- Creates a full image copy if the percentage of changed pages is greater than or equal to *percent_value1*, or if CHANGELIMIT(0) is specified.
- Does not create an image copy if no pages have changed, unless CHANGELIMIT(0) is specified.

If two values are specified, COPY CHANGELIMIT:

- Creates an incremental image copy if the percentage of changed pages is greater than the lowest specified value and less than the highest specified value.
- Creates a full image copy if the percentage of changed pages is equal to or greater than the highest specified value.
- Does not create an image copy if the percentage of changed pages is less than or equal to the lowest specified value.
- If both values are equal, creates a full image copy if the percentage of changed pages is equal to or greater than the specified value.

The **default** values are **(1,10)**.

You cannot use the CHANGELIMIT option for a table space or partition that is defined with TRACKMOD NO. If you change the TRACKMOD option from NO to YES, you must take an image copy before you can use the CHANGELIMIT option. For nonpartitioned table spaces, you must copy the entire table space to allow future CHANGELIMIT requests.

REPORTONLY
Specifies that image copy information is to be displayed. If you specify the REPORTONLY option, only image copy information is displayed. Image copies are not taken in this case; they are only recommended.

DSNUM For a table space, identifies a partition or data set within the table space to be copied; or it copies the entire table space. For an index space, DSNUM identifies a partition to be copied, or it copies the entire index space. This option can specify a partition of a data-partitioned secondary index if the index is copy-enabled.

If a data set of a nonpartitioned table space is in the COPY-pending status, you must copy the entire table space.

ALL Indicates that the entire table space or index space is to be copied. The **default** is **ALL**. You must use ALL for a nonpartitioned secondary index.

integer Is the number of a partition or data set that is to be copied.

An integer value is not valid for nonpartitioned secondary indexes.

For a partitioned table space or index space, the integer is its partition number. The maximum is 4096.

For a nonpartitioned table space, find the integer at the end of the data set name as it is cataloged in the ICF catalog. The data set name has the following format:

```
catname.DSNDBx.dbname.spacename.y0001.Annn
```

In this format:

catname	Is the ICF catalog name or alias.
x	Is C (for VSAM clusters) or D (for VSAM data components).
dbname	Is the database name.
spacename	Is the table space or index space name.
y	Is I or J, which indicates the data set name used by REORG with FASTSWITCH.
nnn	Is the data set integer.

If COPY takes an image copy of data sets (rather than on table spaces), RECOVER, MERGECOPY, or COPYTOCOPY must use the copies on a data set level. For a nonpartitioned table space, if COPY takes image copies on data sets and you run MODIFY RECOVERY with DSNUM ALL, the table space is placed in COPY-pending status if a full image copy of the entire table space does not exist.

PARALLEL Specifies the maximum number of objects in the list that are to be processed in parallel. The utility processes the list of objects in parallel for image copies being written to or from different disk or tape devices. If you specify TAPEUNITS with PARALLEL, you control the number of tape drives that are dynamically allocated for the copy. If you omit PARALLEL, the list is not processed in parallel. See "Copying a list of objects" on page 120 for more information about processing objects in parallel.

Restriction: Do not specify the PARALLEL keyword if one or more of the output data sets are defined with DD statements that specify UNIT=AFF to refer to the same device as a previous DD statement. This usage is not supported with the PARALLEL keyword and could result in an abend. Instead, consider using templates to define your data sets. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

(*num-objects***)**
Specifies the number of objects in the list that are to be processed in parallel. You can adjust this value to a smaller value if COPY encounters storage constraints.

If you specify 0 or do not specify the TAPEUNITS keyword, COPY determines the optimal number of objects to process in parallel.

See "Copying a list of objects" on page 120.

TAPEUNITS Specifies the maximum number of tape drives that the utility dynamically allocates for the list of objects to be processed in parallel. TAPEUNITS applies only to tape drives that are dynamically allocated through the TEMPLATE keyword. It does not apply to JCL allocated tape drives. The total number of tape drives allocated for the COPY request is the sum of the JCL allocated tape drives plus the number of tape drives determined as follows:

- The value specified for TAPEUNITS
- The value determined by the COPY utility if you omit the TAPEUNITS keyword

If you omit this keyword, the utility determines the number of tape drives to dynamically allocate for the copy function.

(*num-tape-units***)**
Specifies the number of tape drives to allocate. If you specify 0 or do not specify a value for *num-tape-units*, COPY determines the maximum number of tape drives to be dynamically allocated for the function.

CHECKPAGE Indicates that each page in the table space or index space is to be checked for validity. The validity checking operates on one page at a time and does not include any cross-page checking. If it finds an error, COPY issues a message that describes the type of error. If more than one error exists in a given page, only the first error is identified. COPY continues checking the remaining pages in the table space or index space after it finds an error.

SYSTEMPAGES
Specifies whether the COPY utility puts system pages at the beginning of the image copy data set.

Although the system pages are located at the beginning of many image copies, this placement is not guaranteed. If system pages have not been changed since the last image copy, then an incremental copy will not include them.

YES Ensures that any header, dictionary, and version system pages are copied at the beginning of the image copy data set. The version system pages can be copied twice.

Selecting YES ensures that the image copy contains the necessary system pages for subsequent UNLOAD utility jobs to correctly format and unload all data rows.

The **default** is **YES**.

NO Does not ensure that the dictionary and version system pages are copied at the beginning of the image copy data set. The COPY utility copies the pages in the current order, including the header pages.

CONCURRENT
Specifies that DFSMSdss concurrent copy is to make the full image

copy. The image copy is recorded in the SYSIBM.SYSCOPY catalog table with ICTYPE=F and STYPE=C or STYPE=J.

If the SYSPRINT DD statement points to a data set, you must use a DSSPRINT DD statement.

When you specify SHRLEVEL(REFERENCE), an ICTYPE=Q record is placed into the SYSIBM.SYSCOPY catalog table after the object has been quiesced. If COPY fails, this record remains in SYSIBM.SYSCOPY. When COPY is successful, this ICTYPE=Q record is replaced with the ICTYPE=F record.

If the page size in the table space matches the control interval for the associated data set, you can use either the SHRLEVEL CHANGE option or the SHRLEVEL REFERENCE option with the CONCURRENT option. If the page size does not match the control interval, you must use the SHRLEVEL REFERENCE option for table spaces with a 8-KB, 16-KB, or 32-KB page size.

When you do not specify FILTERDDN, the DFSMSdss dump statement cannot include more than 255 data sets. When you request a concurrent copy on an object that exceeds this limitation, DB2 dynamically allocates a temporary filter data set for you.

FILTERDDN *ddname*

Specifies the optional DD statement for the filter data set that COPY is to use with the CONCURRENT option. COPY uses this data set to automatically build a list of table spaces that are to be copied by DFSMSdss with one DFSMSdss DUMP statement.

You can use the **FILTERDDN** keyword to specify either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, “TEMPLATE,” on page 593.

If you specify FILTERDDN, the SYSCOPY records for all objects in the list have the same data set name.

ddname is the DD name.

SHRLEVEL Indicates whether other programs can access or update the table space or index while COPY is running.

REFERENCE

Allows read-only access by other programs. The **default** is **REFERENCE**.

CHANGE

Allows other programs to change the table space or index space.

When you specify SHRLEVEL CHANGE, uncommitted data might be copied.

Recommendation: Do not use image copies that are made with SHRLEVEL CHANGE when you run RECOVER TOCOPY.

SHRLEVEL CHANGE is not allowed when you use DFSMSdss concurrent copy for table spaces that have a

page size that is greater than 4 KB and does not match the control interval size. If the page size in the table space matches the control interval size for the associated data set, you can use either the SHRLEVEL CHANGE option or the SHRLEVEL REFERENCE option.

If you are copying a list and you specify the SHRLEVEL CHANGE option, you can specify OPTIONS EVENT(ITEMERROR,SKIP) so that each object in the list is placed in UTRW status and the read claim class is held only while the object is being copied. If you do not specify OPTIONS EVENT(ITEMERROR,SKIP), all of the objects in the list are placed in UTRW status and the read claim class is held on all objects for the entire duration of the COPY.

Instructions for running COPY

To run COPY, you must:

1. Read "Before running COPY" in this section.
2. Prepare the necessary data sets, as described in "Data sets that COPY uses."
3. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for COPY, see "Sample COPY control statements" on page 131.)
4. Prepare a utility control statement, specifying the options for the tasks you want to perform, as described in "Instructions for specific tasks" on page 116.
5. Check the compatibility table in "Concurrency and compatibility for COPY" on page 128 if you want to run other jobs concurrently on the same target objects.
6. Plan for restart if the COPY job doesn't complete, as described in "Terminating or restarting COPY" on page 127.
7. Run COPY by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Before running COPY

Before running COPY, check that the table spaces and index spaces that you want to copy are not in any restricted states. See "Concurrency and compatibility for COPY" on page 128 for a list of restricted states.

Data sets that COPY uses

Table 15 lists the data sets that COPY uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 15. Data sets that COPY uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
DSSPRINT	Output data set for messages when making concurrent copies.	No[1]

Table 15. Data sets that COPY uses (continued)

Data set	Description	Required?
Filter	A single data set that DB2 uses when you specify the FILTERDDN option in the utility control statement. This data set contains a list of VSAM data set names that DB2 builds, and is used during COPY when you specify the CONCURRENT and FILTERDDN options.	No[2]
Copies	From one to four output data sets that contain the resulting image copy data sets. Specify their DD names with the COPYDDN and RECOVERYDDN options of the utility control statement. The default is one copy to be written to the data set described by the SYSCOPY DD statement.	Yes

Notes:

1. Required if you specify CONCURRENT and the SYSPRINT DD statement points to a data set.
2. Required if you specify the FILTERDDN option.

The following objects are named in the utility control statement and do not require DD statements in the JCL:

Table space or index space
Object that is to be copied. (If you want to copy only certain data sets in a table space, you must use the DSNUM option in the control statement.)

DB2 catalog objects
Objects in the catalog that COPY accesses. The utility records each copy in the DB2 catalog table SYSIBM.SYSCOPY.

Output data set size: Image copies are written to sequential non-VSAM data sets.

Recommendation: Use a template for the image copy data set by specifying a TEMPLATE statement without the SPACE keyword. When you omit this keyword, the utility calculates the appropriate size of the data set for you.

Alternatively, you can find the approximate size of the image copy data set for a table space, in bytes, by either executing COPY with the CHANGELIMIT REPORTONLY option, or using the following procedure:

1. Find the high-allocated page number, either from the NACTIVEF column of SYSIBM.SYSTABLESPACE after running the RUNSTATS utility, or from information in the VSAM catalog data set.
2. Multiply the high-allocated page number by the page size.

Filter data set size:

Recommendation: Use a template for the filter data set by specifying a TEMPLATE statement without the SPACE keyword. When you omit this keyword, the utility calculates the appropriate size of the data set for you.

Alternatively, you can determine the approximate size of the filter data set size that is required, in bytes, by using the following formula, where *n* = the number of specified objects in the COPY control statement:

```
(240 + (80 × n))
```

JCL parameters: You can specify a block size for the output by using the BLKSIZE parameter on the DD statement for the output data set. Valid block sizes are multiples of 4096 bytes. You can increase the buffer using the BUFNO parameter; for example, you might specify BUFNO=30, which creates 30 buffers.

See also "Data sets that online utilities use" on page 19 for information about using BUFNO.

Cataloging image copies: To catalog your image copy data sets, use the DISP=(MOD,CATLG,CATLG) parameter in the DD statement or TEMPLATE that is named by the COPYDDN option. After the image copy is taken, the DSVOLSER column of the row that is inserted into SYSIBM.SYSCOPY contains blanks.

Duplicate image copy data sets are not allowed. If a cataloged data set is already recorded in SYSIBM.SYSCOPY with the same name as the new image copy data set, the COPY utility issues a message and does not make the copy.

When RECOVER locates the SYSCOPY entry, it uses the operating system catalog to allocate the required data set. If you have uncataloged the data set, the allocation fails. In that case, the recovery can still go forward; RECOVER searches for a previous image copy. But even if it finds one, RECOVER must use correspondingly more of the log during recovery.

Recommendation: Keep the ICF catalog consistent with the information about existing image copy data sets in the SYSIBM.SYSCOPY catalog table.

Creating the control statement

Create the utility control statement for the COPY job. "Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in your utility control statement:

"Making full image copies" on page 117
"Making incremental image copies" on page 118
"Making multiple image copies" on page 118
"Copying partitions or data sets in separate jobs" on page 120
"Copying a list of objects" on page 120
"Using more than one COPY statement" on page 122
"Copying segmented table spaces" on page 122
"Using DFSMSdss concurrent copy" on page 122
"Specifying conditional image copies" on page 124
"Preparing for recovery" on page 125
"Improving performance" on page 126
"Copying table spaces with mixed volume IDs" on page 126
"Defining generation data groups" on page 126
"Using DB2 with DFSMS products" on page 127
"Putting image copies on tape" on page 127
"Copying a LOB table space" on page 127

Making full image copies

You can make a full image copy of any of the following objects:

- Table space
- Table space partition
- Data set of a linear table space
- Index space
- Index space partition

The following statement specifies that the COPY utility is to make a full image copy of the DSN8S81E table space in database DSN8D81A:

```
COPY TABLESPACE DSN8D81A.DSN8S81E
```

The COPY utility writes pages from the table space or index space to the output data sets. The JCL for the utility job must include DD statements or have a template specification for the data sets. If the object consists of multiple data sets and all are copied in one run, the copies reside in one physical sequential output data set.

Image copies should be made either by entire page set or by partition, but not by both.

Recommendations:

- Take a full image copy after any of the following operations:
  - CREATE or LOAD operations for a new object that is populated.
  - REORG operation for an existing object.
  - LOAD RESUME of an existing object.
- Copy the indexes over a table space whenever a full copy of the table space is taken. More frequent index copies decrease the number of log records that need to be applied during recovery. At a minimum, you should copy an index when it is placed in informational COPY-pending (ICOPY) status. For more information about the ICOPY status, see Appendix C, “Advisory or restrictive states,” on page 853.

If you create an inline copy during LOAD or REORG, you do not need to execute a separate COPY job for the table space. If you do not create an inline copy, and if the LOG option is NO, the COPY-pending status is set for the table space. You must then make a full image copy for any subsequent recovery of the data. An incremental image copy is not allowed in this case.

If the LOG option is YES, the COPY-pending status is not set. However, your next image copy must be a full image copy. Again, an incremental image copy is not allowed.

The COPY utility automatically takes a full image copy of a table space if you attempt to take an incremental image copy when it is not allowed.

The catalog table SYSIBM.SYSCOPY and the directory tables SYSIBM.SYSUTIL and SYSIBM.SYSLGRNX record information from the COPY utility. Copying the catalog table or the directories can lock out separate COPY jobs that are running simultaneously; therefore, defer copying the catalog table or directories until the other copy jobs have completed if possible. However, if you must copy other objects while another COPY job processes catalog tables or directories, specify SHRLEVEL (CHANGE) for the copies of the catalog and directory tables.

Making incremental image copies

An incremental image copy is a copy of the pages that have been changed since the last full or incremental image copy. You cannot take an incremental image copy of an index space. You can make an incremental image copy of a table space if the following conditions are true:

- A full image copy of the table space exists.
- The COPY-pending status is not on for that table space.
- The last copy was taken without the CONCURRENT option.

Copy by partition or data set: You can make an incremental image copy by partition or data set (specified by DSNUM) in the following situations:

- A full image copy of the table space exists.
- A full image copy of the same partition or data set exists and the COPY-pending status is not on for the table space or partition.

In addition, the full image copy must have been made after the most recent use of CREATE, REORG or LOAD, or it must be an inline copy that was made during the most recent use of LOAD or REORG.

Sample control statement: To specify an incremental image copy, use FULL NO on the COPY statement, as in the following example:

```
COPY TABLESPACE DSN8D81A.DSN8S81E
 FULL NO
  SHRLEVEL CHANGE
```

Performance advantage: An incremental image copy generally does not require a complete scan of the table space, with two exceptions:

- The table space is defined with the TRACKMOD NO option.
- You are taking the first copy after you altered a table space to TRACKMOD YES.

Space maps in each table space indicate, for each page, regardless of whether it has changed since the last image copy. Therefore, making an incremental copy can be significantly faster than making a full copy if the table space is defined with the TRACKMOD YES option. Incremental image copies of a table space that is defined with TRACKMOD NO still saves space, although the performance advantage is smaller.

Restrictions: You cannot make incremental copies of DSNDB01.DBD01, DSNDB01.SYSUTILX, or DSNDB06.SYSCOPY in the catalog. For those objects, COPY always makes a full image copy and places the SYSCOPY record in the log.

Making multiple image copies

In a single COPY job, you can create up to four exact copies of any of the following objects:

- Table space
- Table space partition
- Data set of a linear table space
- Index space
- Index space partition

Two copies can be made for use on the local DB2 system (installed with the option LOCALSITE), and two more for offsite recovery (on any system that is installed with the option RECOVERYSITE). All copies are identical, and all are produced at the same time from one invocation of COPY. Alternatively you can use COPYTOCOPY to create the needed image copies. See Chapter 12, “COPYTOCOPY,” on page 143 for more information.

The ICBACKUP column in SYSIBM.SYSCOPY specifies whether the image copy data set is for the local or recovery system, and whether the image copy data set is for the primary copied data set or for the backup copied data set. The ICUNIT column in SYSIBM.SYSCOPY specifies whether the image copy data set is on tape or disk.

Remote-site recovery: For remote site recovery, DB2 assumes that the system and application libraries and the DB2 catalog and directory are identical at the local site and recovery site. You can regularly transport copies of archive logs and database data sets to a safe location to keep current data for remote-site recovery current. This information can be kept on tape until needed.

Naming the data sets for the copies: The COPYDDN option of COPY names the output data sets that receive copies for local use. The RECOVERYDDN option names the output data sets that receive copies that are intended for remote-site recovery. The options have the following formats:

```
COPYDDN (ddname1,ddname2)

RECOVERYDDN (ddname3,ddname4)
```

The DD names for the primary output data sets are *ddname1* and *ddname3*. The ddnames for the backup output data sets are *ddname2* and *ddname4*.

Sample control statement: The following statement makes four full image copies of the table space DSN8S81E in database DSN8D81A. The statement uses LOCALDD1 and LOCALDD2 as DD names for the primary and backup copies that are used on the local system and RECOVDD1 and RECOVDD2 as DD names for the primary and backup copies for remote-site recovery:

```
COPY TABLESPACE DSN8D81A.DSN8S81E
  COPYDDN (LOCALDD1,LOCALDD2)
  RECOVERYDDN (RECOVDD1,RECOVDD2)
```

You do not need to make copies for local use and for remote-site recovery at the same time. COPY allows you to use either the COPYDDN or the RECOVERYDDN option without the other. If you make copies for local use more often than copies for remote-site recovery, a remote-site recovery could be performed with an older copy, and more of the log, than a local recovery; hence, the recovery would take longer. However, in your plans for remote-site recovery, that difference might be acceptable. You can also use MERGECOPY RECOVERYDDN to create recovery-site full image copies, and merge local incremental copies into new recovery-site full copies.

Conditions for making multiple incremental image copies: DB2 cannot make incremental image copies if any of the following conditions is true:

- The incremental image copy is requested only for a site other than the current site (the local site from which the request is made).
- Incremental image copies are requested for both sites, but the most recent full image copy was made for only one site.
- Incremental image copies are requested for both sites and the most recent full image copies were made for both sites, but between the most recent full image copy and current request, incremental image copies were made for the current site only.

If you attempt to make incremental image copies under any of these conditions, COPY terminates with return code 8, does not take the image copy or update the SYSIBM.SYSCOPY table, and issues the following message:

```
DSNU404I  csect-name
 LOCAL SITE AND RECOVERY SITE INCREMENTAL
          IMAGE COPIES ARE NOT SYNCHRONIZED
```

To proceed, and still keep the two sets of data synchronized, take another full image copy of the table space for both sites, or change your request to make an incremental image copy only for the site at which you are working.

DB2 cannot make an incremental image copy if the object that is being copied is an index or index space.

Maintaining copy consistency: Make full image copies for both the local and recovery sites:
- If a table space is in COPY-pending status
- After a LOAD or REORG procedure that did not create an inline copy
- If an index is in the informational COPY-pending status

This action helps to ensure correct recovery for both local and recovery sites. If the requested full image copy is for one site only, but the history shows that copies were made previously for both sites, COPY continues to process the image copy and issues the following warning message:

```
DSNU406I  FULL IMAGE COPY SHOULD BE TAKEN FOR BOTH LOCAL SITE AND
          RECOVERY SITE.
```

The COPY-pending status of a table space is not changed for the other site when you make multiple image copies at the current site for that other site. For example, if a table space is in COPY-pending status at the current site, and you make copies from there for the other site only, the COPY-pending status is still on when you bring up the system at that other site.

Copying partitions or data sets in separate jobs

If you have a partitioned table space or partitioning index, you can copy the partitions independently in separate simultaneous jobs. This can reduce the time it takes to create an image copy of the total table space.

If a nonpartitioned table space consists of more than one data set, you can copy several or all of the data sets independently in separate jobs. To do so, run simultaneous COPY jobs (one job for each data set) and specify SHRLEVEL CHANGE on each job.

However, creating copies simultaneously does not provide you with a consistent recovery point unless you subsequently run a QUIESCE for the table space.

Copying a list of objects

Within a single COPY control statement, the COPY utility allows you to process a list that contains any of the following objects:
- Table space
- Table space partition
- Data set of a linear table space
- Index space
- Index space partition

Specifying objects in a list is useful for copying a complete set of referentially related table spaces before running QUIESCE. Consider the following information when taking an image copy for a list of objects:

- DB2 copies table spaces and index spaces in the list one at a time, in the specified order, unless you invoke parallelism by specifying the PARALLEL keyword.
- Each table space in the list with a CHANGELIMIT specification has a REPORT phase, so the phase switches between REPORT and COPY while processing the list.
- If processing completes successfully, any COPY-pending status on the table spaces and informational COPY-pending status on the indexes are reset.
- If you use COPY with the SHRLEVEL(REFERENCE) option:
  - DB2 drains the write claim class on each table space and index in the UTILINIT phase, which is held for the duration of utility processing.
  - Utility processing inserts SYSCOPY rows for all of the objects in the list at the same time, after all of the objects have been copied.
  - All objects in the list have identical RBA or LRSN values for the START_RBA column for the SYSCOPY rows: the START_RBA is set to the current LRSN at the end of the COPY phase.
- If you use COPY with the SHRLEVEL(CHANGE) option:
  - If you specify OPTIONS EVENT(ITEMERROR,SKIP), each object in the list is placed in UTRW status and the read claim class is held only while the object is being copied. If you do not specify OPTIONS EVENT(ITEMERROR,SKIP), all of the objects in the list are placed in UTRW status and the read claim class is held on all objects for the entire duration of the COPY.
  - Utility processing inserts a SYSCOPY row for each object in the list when the copy of each object is complete.
  - Objects in the list have different LRSN values for the START_RBA column for the SYSCOPY rows; the START_RBA value is set to the current RBA or LRSN at the start of copy processing for that object.

When you specify the PARALLEL keyword, DB2 supports parallelism for image copies on disk or tape devices. You can control the number of tape devices to allocate for the copy function by using TAPEUNITS with the PARALLEL keyword. If you use JCL statements to define tape devices, the JCL controls the allocation of the devices.

When you explicitly specify objects with the PARALLEL keyword, the objects are not necessarily processed in the specified order. Objects that are to be written to tape and whose file sequence numbers have been specified in the JCL are processed in the specified order. If templates are used, you cannot specify file sequence numbers. In the absence of overriding JCL specifications, DB2 determines the placement and, thus, the order of processing for such objects. When only templates are used, objects are processed according to their size, with the largest objects processed first.

Both the PARALLEL and TAPEUNITS keywords act as constraints on the processing of the COPY utility. The PARALLEL keyword constrains the amount of parallelism by restricting the maximum number of objects that can be processed simultaneously. The TAPEUNITS keyword constrains the number of tape drives that can be dynamically allocated for the COPY command. The TAPEUNITS keyword can constrain the amount of parallelism if an object requires a number of tapes such that the number of remaining tapes is insufficient to service another object.

To calculate the number of threads that you need when you specify the PARALLEL keyword, use the formula (*n* * 2 + 1), where *n* is the number of objects that are to be processed in parallel, regardless of the total number of objects in the list. If you do not use the PARALLEL keyword, *n* is 1 and COPY uses three threads for a single-object COPY job.

The following table spaces cannot be included in a list of table spaces. You must specify each one as a single object:
- DSNDB01.SYSUTILX
- DSNDB06.SYSCOPY
- DSNDB01.SYSLGRNX

The only exceptions to this restriction are the indexes over these table spaces that were defined with the COPY YES attribute. You can specify such indexes along with the appropriate table space.

Using more than one COPY statement

You can use more than one control statement for COPY in one DB2 utility job step. After each COPY statement executes successfully:
- A row that refers to each image copy is recorded in the SYSIBM.SYSCOPY table.
- The image copy data sets are valid and available for RECOVER, MERGECOPY, COPYTOCOPY, and UNLOAD.

If a job step that contains more than one COPY statement abends, **do not use TERM UTILITY.** Restart the job from the last commit point by using RESTART instead. Terminating COPY by using TERM UTILITY in this case creates inconsistencies between the ICF catalog and DB2 catalogs.

Copying segmented table spaces

COPY distinguishes between segmented and nonsegmented table spaces. If you specify a segmented table space, COPY locates empty and unformatted data pages in the table space and does not copy them.

Using DFSMSdss concurrent copy

You might be able to gain improved availability by using the concurrent copy function of the DFSMSdss component of the Data Facility Storage Management Subsystem (DFSMS). You can subsequently run the DB2 RECOVER utility to restore those image copies and apply the necessary log records to them to complete recovery.

The CONCURRENT option of COPY invokes DFSMSdss concurrent copy. The COPY utility records the resulting DFSMSdss concurrent copies in the catalog table SYSIBM.SYSCOPY with ICTYPE=F and STYPE=C or STYPE=J. STYPE=C indicates that the concurrent copy was taken of the ″I″ instance of the table space (which means that the instance qualifier in the name of the corresponding data set begins with the letter ″I″). STYPE=J indicates that the concurrent copy was taken of the ″J″ instance of the table space (which means that the instance qualifier in the name of the corresponding data set begins with the letter ″J″). See “Shadow data sets” on page 457 for an explanation of data set naming conventions used by REORG SHRLEVEL CHANGE.

To obtain a consistent offline backup copy outside of DB2:

1. Start the DB2 objects that are being backed up for read-only access by issuing the following command:

   ```
   -START DATABASE(database-name) SPACENAM(
   tablespace-name) ACCESS(RO)
   ```

Allowing read-only access is necessary to ensure that no updates to data occur during this procedure.

2. Run QUIESCE with the WRITE(YES) option to quiesce all DB2 objects that are being backed up.
3. Back up the DB2 data sets after the QUIESCE utility completes successfully.
4. Issue the following command to allow transactions to access the data:

   ```
   -START DATABASE(database-name) SPACENAM(tablespace-name)
   ```

If you use the CONCURRENT option:

- You must supply either a COPYDDN DD name, a RECOVERYDDN DD name, or both.
- You can set the disposition to DISP=(MOD,CATLG,CATLG) if you specify the new data set for the image copy on a scratch volume (a specific volume serial number is not specified). You must set the disposition to DISP=(NEW,CATLG,CATLG) if you specify a specific volume serial number for the new image copy data set.
- If you are restarting COPY, specify DISP=(MOD,CATLG,CATLG) or DISP=(NEW,CATLG,CATLG) for the COPYDDN and RECOVERYDDN data sets. The DFSMSdss DUMP command does not support appending to an existing data set. Therefore, the COPY utility converts any DISP=MOD data sets to DISP=OLD before invoking DFSMSdss.
- If the SYSPRINT DD statement points to a data set, you must use a DSSPRINT DD statement.
- If the page size in the table space matches the control interval for the associated data set, you can use either the SHRLEVEL CHANGE option or the SHRLEVEL REFERENCE option. If the page size does not match the control interval, you must use the SHRLEVEL REFERENCE option for table spaces with a 8-KB, 16-KB, or 32-KB page size.

Restrictions on using DFSMSdss concurrent copy: You cannot use a copy that is made with DFSMSdss concurrent copy with the PAGE or ERRORRANGE options of the RECOVER utility. If you specify PAGE or ERROR RANGE, RECOVER bypasses any concurrent copy records when searching the SYSIBM.SYSCOPY table for a recovery point.

You can use the CONCURRENT option with SHRLEVEL CHANGE on a table space if the page size in the table space matches the control interval for the associated data set.

Also, you cannot run the following DB2 stand-alone utilities on copies that are made by DFSMSdss concurrent copy:

DSN1COMP
DSN1COPY
DSN1PRNT

You cannot execute the CONCURRENT option from the DB2I Utilities panel or from the DSNU TSO CLIST command.

Requirements for using DFSMSdss concurrent copy: DFSMSdss concurrent copy is enabled by specific hardware. Contact IBM or the vendor for your specific storage product to verify whether your controller or storage server supports the concurrent copy function.

Table space availability: If you specify COPY SHRLEVEL REFERENCE with the CONCURRENT option, and if you want to copy all of the data sets for a list of table spaces to the same dump data set, specify FILTERDDN in your COPY statement to improve table space availability. If you do not specify FILTERDDN, COPY might force DFSMSdss to process the list of table spaces sequentially, which might limit the availability of some of the table spaces that are being copied.

Specifying conditional image copies

Use the CHANGELIMIT option of the COPY utility to specify conditional image copies. You can use it to get a report of image copy information about a table space, or you can let DB2 decide whether to take an image copy based on this information.

You cannot use the CHANGELIMIT option for a table space or partition that is defined with TRACKMOD NO. If you change the TRACKMOD option from NO to YES, you must take an image copy before you can use the CHANGELIMIT option. When you change the TRACKMOD option from NO to YES for a linear table space, you must take a full image copy by using DSNUM ALL before you can copy using the CHANGELIMIT option.

Obtaining image copy information about a table space: When you specify COPY CHANGELIMIT REPORTONLY, COPY reports image copy information for the table space and recommends the type of copy, if any, to take. The report includes:

- The total number of pages in the table space. This value is the number of pages that are to be copied if a full image copy is taken.
- The number of empty pages, if the table space is segmented.
- The number of changed pages. This value is the number of pages that are to be copied if an incremental image copy is taken.
- The percentage of changed pages.
- The type of image copy that is recommended.

Adding conditional code to your COPY job: You can add conditional code to your jobs so that an incremental or full image copy, or some other step, is performed depending on how much the table space has changed. For example, you can add a conditional MERGECOPY step to create a new full image copy if your COPY job took an incremental copy. COPY CHANGELIMIT uses the following return codes to indicate the degree that a table space or list of table spaces has changed:

1 (informational)
If no CHANGELIMIT was met.

2 (informational)
If the percentage of changed pages is greater than the low CHANGELIMIT and less than the high CHANGELIMIT value.

3 (informational)
If the percentage of changed pages is greater than or equal to the high CHANGELIMIT value.

If you specify multiple COPY control statements in one job step, that job step reports the highest return code from all of the imbedded statements. Basically, the statement with the highest percentage of changed pages determines the return code and the recommended action for the entire list of COPY control statements that are contained in the subsequent job step.

Using conditional copy with generation data groups (GDGs): When you use generation data groups (GDGs) and need to make an incremental image copy, take the following steps to prevent creating an empty image copy:

1. Include in your job a first step in which you run COPY with CHANGELIMIT REPORTONLY. Set the SYSCOPY DD statement to DD DUMMY so that no output data set is allocated. If you specify REPORTONLY and use a template, DB2 does not dynamically allocate the data set.
2. Add a conditional JCL statement to examine the return code from the COPY CHANGELIMIT REPORTONLY step.
3. Add a second COPY step without CHANGELIMIT REPORTONLY to copy the table space or table space list based on the return code from the second step.

Preparing for recovery

Read the following topics pertaining to recovery, if you are taking incremental copies, if you have recently run REORG or LOAD, or if you plan to recover a LOB table space.

Using incremental copies: The RECOVER TABLESPACE utility merges all incremental image copies since the last full image copy, and it must have all the image copies available at the same time. If this requirement is likely to strain your system resources—for example, by demanding more tape units than are available—consider regularly merging multiple image copies into one copy.

Even if you do not periodically merge multiple image copies into one copy when you do not have enough tape units, RECOVER TABLESPACE can still attempt to recover the object. RECOVER dynamically allocates the full image copy and attempts to dynamically allocate all the incremental image copy data sets. If every incremental copy can be allocated, recovery proceeds to merge pages to table spaces and apply the log. If a point is reached where RECOVER TABLESPACE cannot allocate an incremental copy, the log RBA of the last successfully allocated data set is noted. Attempts to allocate incremental copies cease, and the merge proceeds using only the allocated data sets. The log is applied from the noted RBA, and the incremental image copies that were not allocated are simply ignored.

After running LOAD or REORG: **Recommendation:** Create primary and backup image copies after specifying a LOAD or REORG operation with LOG NO when an inline copy is not created. Create these copies, so that if the primary image copy is not available, fallback recovery using the secondary image copy is possible.

Creating a point of recovery: If you use COPY SHRLEVEL REFERENCE to copy a list of objects that contains all referentially related structures, you do not need to QUIESCE these objects in order to create a consistent point of recovery.

For LOB data, you should quiesce and copy both the base table space and the LOB table space at the same time to establish a recovery point of consistency, called a recovery point. Be aware that QUIESCE does not create a recovery point for a LOB table space that contains LOBs that are defined with LOG NO.

Setting and clearing the informational COPY-pending status: For an index that was defined with the COPY YES attribute the following utilities can place the index in the informational COPY-pending (ICOPY) status:

- REORG INDEX
- REORG TABLESPACE LOG YES or NO
- LOAD TABLE LOG YES or NO
- REBUILD INDEX

After the utility processing completes, take a full image copy of the index space so that the RECOVER utility can recover the index space. If you need to recover an index of which you did not take a full image copy, use the REBUILD INDEX utility to rebuild the index from data in the table space.

Improving performance

You can merge a full image copy and subsequent incremental image copies into a new full copy by running the MERGECOPY utility. After reorganizing a table space, the first image copy **must** be a full image copy.

Do not base the decision of whether to run a full image copy or an incremental image copy on the number of rows that are updated since the last image copy was taken. Instead, base your decision on the percentage of pages that contain at least one updated record (not the number of updated records). Regardless of the size of the table, if more than 50% of the pages contain updated records, use full image copy (this saves the cost of a subsequent MERGECOPY). To find the percentage of changed pages, you can execute COPY with the CHANGELIMIT REPORTONLY option. Alternatively, you can execute COPY CHANGELIMIT to allow COPY to determine whether a full image copy or incremental copy is required; see "Specifying conditional image copies" on page 124 for more information.

Using data compression can improve COPY performance because COPY does not decompress data. The performance improvement is proportional to the amount of compression.

Copying table spaces with mixed volume IDs

You cannot copy a table space or index space that uses a storage group that is defined with mixed specific and non-specific volume IDs by using CREATE STOGROUP or ALTER STOGROUP. If you specify such a table space or index space, the job terminates and you receive error message DSNU419I.

Defining generation data groups

Recommendation: Use generation data groups to hold image copies, because their use automates the allocation of data set names and the deletion of the oldest data set. When you define the generation data group:

- You can specify that the oldest data set is automatically deleted when the maximum number of data sets is reached. If you do that, make the maximum number large enough to support all recovery requirements. When data sets are deleted, use the MODIFY utility to delete the corresponding rows in SYSIBM.SYSCOPY.
- Make the limit number of generation data sets equal to the number of copies that you want to keep. Use NOEMPTY to avoid deleting all the data sets from the integrated catalog facility catalog when the limit is reached.

Attention: Do not take incremental image copies when using generation data groups unless data pages have changed. When you use generation data groups, taking an incremental image copy when no data pages have changed causes the following results:

- The new image copy data set is empty.
- No SYSCOPY record is inserted for the new image copy data set.
- Your oldest image copy is deleted.

See "Using conditional copy with generation data groups (GDGs)" on page 125 for guidance on executing COPY with the CHANGELIMIT and REPORTONLY options to ensure that you do not create empty image copy data sets when using GDGs.

Recommendation: Use templates when using generation data groups.

Using DB2 with DFSMS products

If image copy data sets are managed by HSM or SMS, all data sets are cataloged.

If you plan to use SMS, catalog all image copies. Never maintain cataloged and uncataloged image copies that have the same name.

Putting image copies on tape

Do not combine a full image copy and incremental image copies for the same table space on one tape volume. If you do, the RECOVER TABLESPACE utility cannot allocate the incremental image copies.

Copying a LOB table space

Both full and incremental image copies are supported for a LOB table space, as well as SHRLEVEL REFERENCE, SHRLEVEL CHANGE, and the CONCURRENT options. COPY without the CONCURRENT option does not copy empty or unformatted data pages of a LOB table space.

Terminating or restarting COPY

This section explains the tasks of terminating and restarting the COPY utility.

Terminating COPY

This section explains the recommended way to terminate the COPY utility.

An active or stopped COPY job may be terminated with the TERM UTILITY command. However, if you issue TERM UTILITY while COPY is in the active or stopped state, DB2 inserts an ICTYPE=T record in the SYSIBM.SYSCOPY catalog table for each object that COPY had started processing, but not yet completed. (Exception: If the COPY utility is already in the UTILTERM phase, the image copy is considered completed.) For copies that are made with SHRLEVEL REFERENCE, some objects in the list might not have an ICTYPE=T record. For SHRLEVEL CHANGE, some objects might have a valid ICTYPE=F, I, or T record, or no record at all. The COPY utility does not allow you to take an incremental image copy if an ICTYPE=T record exists. You must make a full image copy.

Implications of DISP on the DD statement: The result of terminating a COPY job that uses the parameter DISP=(MOD,CATLG,CATLG) varies as follows:

- If only one COPY control statement exists, no row is written to SYSIBM.SYSCOPY, but an image copy data set has been created and is cataloged in the ICF catalog. You should delete that image copy data set.
- If several COPY control statements are in one COPY job step, a row for each successfully completed copy is written to SYSIBM.SYSCOPY. However, all the image copy data sets have been created and cataloged. You should delete all image copy data sets that are not recorded in SYSIBM.SYSCOPY.

Restarting COPY

Recommendation: Use restart current instead, because it:

- Is valid for full image copies and incremental copies
- Is valid for a single job step with several COPY control statements
- Is valid for a list of objects
- Requires a minimum of re-processing
- Keeps the DB2 catalog and the integrated catalog facility catalog synchronized

If you do **not** use the TERM UTILITY command, you can restart a COPY job. COPY jobs with the CONCURRENT option restart from the beginning, and other COPY jobs restart from the last commit point. You cannot use RESTART(PHASE) for any COPY job. If you are restarting a COPY job with uncataloged output data sets, you must specify the appropriate volumes for the job in the JCL or on the TEMPLATE utility statement. Doing so could impact your ability to use implicit restart. For general instructions on restarting a utility job, see "Restarting an online utility" on page 43.

Restarting with a new data set: If you define a new output data set for a current restart, complete the following actions before restarting the COPY job:

1. Copy the failed COPY output to the new data set.
2. Delete the old data set.
3. Rename the new data set to use the old data set name.

Restarting COPY after an out-of-space condition: If the recommended procedure is not followed an ABEND 413-1C may occur during restart of the COPY.

See "Restarting after the output data set is full" on page 45 for guidance in restarting COPY from the last commit point after receiving an out-of-space condition.

Concurrency and compatibility for COPY

DB2 treats individual data and index partitions as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible.

Restricted states: Do not copy a table space that is in any of the following states:
- CHECK-pending
- RECOVER-pending
- REFRESH-pending
- Logical error range
- Group buffer pool RECOVER-pending
- Stopped
- STOP-pending

Do not copy an index space that is in any of the following states:
- CHECK-pending
- REBUILD-pending
- RECOVER-pending
- REFRESH pending
- Logical error range
- Group buffer pool RECOVER-pending
- Stopped
- STOP-pending

See Appendix C, "Advisory or restrictive states," on page 853 for information about resetting CHECK-pending, REBUILD-pending, RECOVER-pending, REFRESH-pending, and group buffer pool RECOVER-pending statuses. See the description of message DSNU205I in *DB2 Messages* for information about correcting the logical error range. See "Determining the status of a utility" on page 39 for information about resetting the stopped status.

If a table space is in COPY-pending status, or an index is in informational COPY-pending status, you can reset the status only by taking a full image copy of the entire table space, all partitions of the partitioned table space, or the index space. When you make an image copy of a partition, the COPY-pending status of the partition is reset. If a nonpartitioned table space is in COPY-pending status, you can reset the status only by taking a full image copy of the entire table space, and not of each data set.

Claims and drains: Table 16 shows which claim classes COPY claims and drains and any restrictive status that the utility sets on the target object.

Table 16. Claim classes of COPY operations

Target	SHRLEVEL REFERENCE	SHRLEVEL CHANGE
Table space, index space, or partition	DW UTRO	CR UTRW[1]

Legend:
- DW - Drain the write claim class - concurrent access for SQL readers
- CR - Claim the read claim class
- UTRO - Utility restrictive state, read-only access allowed
- UTRW - Utility restrictive state, read-write access allowed

Notes:

1. If the target object is a segmented table space, SHRLEVEL CHANGE does not allow you to concurrently execute an SQL DELETE without the WHERE clause.

COPY does not set a utility restrictive state if the target object is DSNDB01.SYSUTILX.

Compatibility: Table 17 documents which utilities can run concurrently with COPY on the same target object. The target object can be a table space, an index space, or a partition of a table space or index space. If compatibility depends on particular options of a utility, that information is also documented in the table.

Table 17. Compatibility of COPY with other utilities

Action	COPY INDEXSPACE SHRLEVEL REFERENCE	COPY INDEXSPACE SHRLEVEL CHANGE	COPY TABLESPACE SHRLEVEL REFERENCE[1]	COPY TABLESPACE SHRLEVEL CHANGE
BACKUP SYSTEM	Yes	Yes	Yes	Yes
CHECK DATA	Yes	Yes	No	No
CHECK INDEX	Yes	Yes	Yes	Yes
CHECK LOB	Yes	Yes	Yes	Yes
COPY INDEXSPACE	No	No	Yes	Yes
COPY TABLESPACE	Yes	Yes	No	No
COPYTOCOPY	No	No	No	No
DIAGNOSE	Yes	Yes	Yes	Yes
LOAD	No	No	No	No
MERGECOPY	No	No	No	No
MODIFY	No	No	No	No

Table 17. Compatibility of COPY with other utilities (continued)

Action	COPY INDEXSPACE SHRLEVEL REFERENCE	COPY INDEXSPACE SHRLEVEL CHANGE	COPY TABLESPACE SHRLEVEL REFERENCE[1]	COPY TABLESPACE SHRLEVEL CHANGE
QUIESCE	Yes	No	Yes	No
REBUILD INDEX	No	No	Yes	Yes
RECOVER INDEX	No	No	Yes	Yes
RECOVER TABLESPACE	Yes	Yes	No	No
REORG INDEX	No	No	Yes	Yes
REORG TABLESPACE UNLOAD CONTINUE or PAUSE	No	No	No	No
REORG TABLESPACE UNLOAD ONLY or EXTERNAL	Yes	Yes	Yes	Yes
REPAIR LOCATE by KEY, RID, or PAGE DUMP or VERIFY	Yes	Yes	Yes	Yes
REPAIR LOCATE by KEY or RID DELETE or REPLACE	No	No	No	No
REPAIR LOCATE INDEX PAGE REPLACE	No	No	Yes	No
REPAIR LOCATE TABLESPACE PAGE REPLACE	Yes	Yes	No	No
REPORT	Yes	Yes	Yes	Yes
RESTORE SYSTEM	No	No	No	No
RUNSTATS INDEX	Yes	Yes	Yes	Yes
RUNSTATS TABLESPACE	Yes	Yes	Yes	Yes
STOSPACE	Yes	Yes	Yes	Yes
UNLOAD[1]	Yes	Yes	Yes	Yes

Notes:

1. If CONCURRENT option is used, contention might be encountered when other utilities are run on the same object at the same time.

To run on DSNDB01.SYSUTILX, COPY must be the only utility in the job step. Also, if SHRLEVEL REFERENCE is specified, the COPY job of DSNDB01.SYSUTILX must be the only utility running in the Sysplex.

COPY on SYSUTILX is an “exclusive” job; such a job can interrupt another job between job steps, possibly causing the interrupted job to time out.

Sample COPY control statements

In some cases, you might run a COPY utility job more than once. To avoid duplicate image copy data sets, a DSN qualifier is used in the following examples. See the description of the COPYDDN parameter in "Option descriptions" on page 107 for further information.

Example 1: Making a full image copy. The following control statement specifies that the COPY utility is to make a full image copy of table space DSN8D81A.DSN8S81E. The copy is to be written to the data set that is defined by the SYSCOPY DD statement in the JCL; SYSCOPY is the default.

```
//STEP1    EXEC DSNUPROC,UID='IUJMU111.COPYTS',
//         UTPROC='',
//         SYSTEM='DSN',DB2LEV=DB2A
//SYSCOPY  DD DSN=COPY001F.IFDY01,UNIT=SYSDA,VOL=SER=CPY01I,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//SYSIN DD *
COPY TABLESPACE DSN8D81A.DSN8S81E
/*
```

Instead of defining the data sets in the JCL, you can use templates. In the following example, the preceding job is modified to use a template. In this example, the name of the template is LOCALDDN. The LOCALDDN template is identified in the COPY statement by the COPYDDN option.

```
//STEP1    EXEC DSNUPROC,UID='IUJMU111.COPYTS',
//         UTPROC='',
//         SYSTEM='DSN',DB2LEV=DB2A
//SYSIN DD *
TEMPLATE LOCALDDN UNIT SYSDA DSN(COPY001F.IFDY01)
          SPACE(15,1) CYL DISP(NEW,CATLG,CATLG)
 COPY TABLESPACE DSN8D81A.DSN8S81E COPYDDN(LOCALDDN)
/*
```

Recommendation: When possible, use templates to allocate data sets. For more information about templates, see Chapter 31, "TEMPLATE," on page 593.

Example 2: Making full image copies for local site and recovery site. The following COPY control statement specifies that COPY is to make primary and backup full image copies of table space DSN8D81P.DSN8S81C at both the local site and the recovery site. The COPYDDN option specifies the output data sets for the local site, and the RECOVERYDDN option specifies the output data sets for the recovery site. The PARALLEL option indicates that up to 2 objects are to be processed in parallel.

The OPTIONS statement at the beginning indicates that if COPY encounters any errors (return code 8) while making the requested copies, DB2 ignores that particular item. COPY skips that item and moves on to the next item. For example, if DB2 encounters an error copying the specified data set to the COPY1 data set, DB2 ignores the error and tries to copy the table space to the COPY2 data set.

```
OPTIONS EVENT(ITEMERROR,SKIP)
COPY TABLESPACE DSN8D81P.DSN8S81C
   COPYDDN(COPY1,COPY2)
   RECOVERYDDN(COPY3,COPY4)
   PARALLEL(2)
```

Example 3: Making full image copies of a list of objects. The control statement in Figure 20 on page 133 specifies that COPY is to make local and recovery full image copies (both primary and backup) of the following objects:

- Table space DSN8D81A.DSN8S81D, and its indexes:
  - DSN8810.XDEPT1
  - DSN8810.XDEPT2
  - DSN8810.XDEPT3
- Table space DSN8D81A.DSN8S81E, and its indexes:
  - DSN8710.XEMP1
  - DSN8710.XEMP2

These copies are to be written to the data sets that are identified by the COPYDDN and RECOVERYDDN options for each object. The COPYDDN option specifies the data sets for the copies at the local site, and the RECOVERYDDN option specifies the data sets for the copies at the recovery site. The first parameter of each of these options specifies the data set for the primary copy, and the second parameter specifies the data set for the backup copy. For example, the primary copy of table space DSN8D81A.DSN8S81D at the recovery site is to be written to the data set that is identified by the COPY3 DD statement.

PARALLEL(4) indicates that up to four of these objects can be processed in parallel. As the COPY job of an object completes, the next object in the list begins processing in parallel until all of the objects have been processed.

SHRLEVEL REFERENCE specifies that no updates are allowed during the COPY job. This option is the default and is recommended to ensure the integrity of the data in the image copy.

```
//STEP1    EXEC DSNUPROC,UID='IUJMU111.COPYTS',
//         UTPROC='',
//         SYSTEM='DSN',DB2LEV=DB2A
//COPY1 DD DSN=C81A.S20001.D2003142.T155241.LP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY2 DD DSN=C81A.S20001.D2003142.T155241.LB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY3 DD DSN=C81A.S20001.D2003142.T155241.RP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY4 DD DSN=C81A.S20001.D2003142.T155241.RB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY5 DD DSN=C81A.S20002.D2003142.T155241.LP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY6 DD DSN=C81A.S20002.D2003142.T155241.LB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY7 DD DSN=C81A.S20002.D2003142.T155241.RP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY8 DD DSN=C81A.S20002.D2003142.T155241.RB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY1 DD DSN=C81A.S20001.D2003142.T155241.LP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY2 DD DSN=C81A.S20001.D2003142.T155241.LB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY3 DD DSN=C81A.S20001.D2003142.T155241.RP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY4 DD DSN=C81A.S20001.D2003142.T155241.RB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY5 DD DSN=C81A.S20002.D2003142.T155241.LP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY6 DD DSN=C81A.S20002.D2003142.T155241.LB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY7 DD DSN=C81A.S20002.D2003142.T155241.RP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY8 DD DSN=C81A.S20002.D2003142.T155241.RB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY9 DD DSN=C81A.S20003.D2003142.T155241.LP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY10 DD DSN=C81A.S20003.D2003142.T155241.LB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY11 DD DSN=C81A.S20003.D2003142.T155241.RP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY12 DD DSN=C81A.S00003.D2003142.T155241.RB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY13 DD DSN=C81A.S00004.D2003142.T155241.LP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY14 DD DSN=C81A.S00004.D2003142.T155241.LB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY15 DD DSN=C81A.S00004.D2003142.T155241.RP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
```

Figure 20. Example of making full image copies of multiple objects (Part 1 of 2)

```
//COPY16 DD DSN=C81A.S00004.D2003142.T155241.RB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY17 DD DSN=C81A.S00005.D2003142.T155241.LP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY18 DD DSN=C81A.S00005.D2003142.T155241.LB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY19 DD DSN=C81A.S00005.D2003142.T155241.RP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY20 DD DSN=C81A.S00005.D2003142.T155241.RB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY21 DD DSN=C81A.S00006.D2003142.T155241.LP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY22 DD DSN=C81A.S00006.D2003142.T155241.LB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY23 DD DSN=C81A.S00006.D2003142.T155241.RP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY24 DD DSN=C81A.S00006.D2003142.T155241.RB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY25 DD DSN=C81A.S00007.D2003142.T155241.LP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY26 DD DSN=C81A.S00007.D2003142.T155241.LB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY27 DD DSN=C81A.S00007.D2003142.T155241.RP,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY28 DD DSN=C81A.S00007.D2003142.T155241.RB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//SYSIN DD *
 COPY
   TABLESPACE DSN8D81A.DSN8S81D
        COPYDDN (COPY1,COPY2)
        RECOVERYDDN (COPY3,COPY4)
  INDEX DSN8810.XDEPT1
        COPYDDN (COPY5,COPY6)
        RECOVERYDDN (COPY7,COPY8)
  INDEX DSN8810.XDEPT2
        COPYDDN (COPY9,COPY10)
        RECOVERYDDN (COPY11,COPY12)
  INDEX DSN8810.XDEPT3
        COPYDDN (COPY13,COPY14)
        RECOVERYDDN (COPY15,COPY16)
  TABLESPACE DSN8D81A.DSN8S81E
        COPYDDN (COPY17,COPY18)
        RECOVERYDDN (COPY19,COPY20)
  INDEX DSN8810.XEMP1
        COPYDDN (COPY21,COPY22)
        RECOVERYDDN (COPY23,COPY24)
  INDEX DSN8810.XEMP2
        COPYDDN (COPY25,COPY26)
        RECOVERYDDN (COPY27,COPY28)
 PARALLEL(4)
 SHRLEVEL REFERENCE
/*
```

Figure 20. Example of making full image copies of multiple objects (Part 2 of 2)

You can also write this COPY job so that it uses lists and templates, as shown in Figure 21 on page 135. In this example, the name of the template is COPY. Note that this TEMPLATE statement does not contain any space specifications for the dynamically allocated data sets. Instead, DB2 determines the space requirements. The COPY template is identified in the COPY statement by the COPYDDN and RECOVERYDDN options. The name of the list is COPYLIST. This list is identified in the COPY control statement by the LIST option.

```
//STEP1    EXEC DSNUPROC,UID='IUJMU111.COPYTS',
//         UTPROC='',
//         SYSTEM='DSN',DB2LEV=DB2A
//SYSIN DD *
TEMPLATE COPY UNIT SYSDA
   DSN 'C&DB(6)..S&SEQ..D&DATE..T&TIME..&LR.&PB.'
            CYL DISP(NEW,CATLG,CATLG)
  LISTDEF COPYLIST
          INCLUDE TABLESPACE DSN8D81A.DSN8S81D
          INCLUDE INDEX DSN8810.XDEPT1
          INCLUDE INDEX DSN8810.XDEPT2
          INCLUDE INDEX DSN8810.XDEPT3
          INCLUDE TABLESPACE DSN8D81A.DSN8S81E
          INCLUDE INDEX DSN8810.XEMP1
          INCLUDE INDEX DSN8810.XEMP2
   COPY LIST COPYLIST COPYDDN(COPY,COPY)
         RECOVERYDDN(COPY,COPY)
     PARALLEL(4) SHRLEVEL REFERENCE
/*
```

Figure 21. Example of using a list and template to make full image copies of multiple objects

Note that the DSN option of the TEMPLATE statement identifies the names of the data sets to which the copies are to be written. These names are similar to the data set names in the JCL in Figure 20 on page 133. For more information about using variable notation for data set names in TEMPLATE statements, see "Creating data set names" on page 607.

Each of the preceding COPY jobs create a point of consistency for the table spaces and their indexes. You can subsequently use the RECOVER utility with the TOLOGPOINT option to recover all of these objects; see 387 for an example.

Example 4: Making full image copies of a list of objects in parallel on tape. The following COPY control statement specifies that COPY is to make image copies of the specified table spaces and their associated index spaces in parallel and stack the copies on different tape devices.

The PARALLEL 2 option specifies that up to two objects can be processed in parallel. The TAPEUNITS 2 option specifies that up to two tape devices can be dynamically allocated at one time. The COPYDDN option for each object specifies the data set that is to be used for the local image copy. In this example, all of these data sets are dynamically allocated and defined by templates. For example, table space DSN8D81A.DSN8S81D is copied into a data set that is defined by the A1 template.

The TEMPLATE utility control statements define the templates A1 and A2. For more information about TEMPLATE control statements, see "Syntax and options of the TEMPLATE control statement " on page 593 in the TEMPLATE chapter.

```
//COPY2A   EXEC DSNUPROC,SYSTEM=DSN
//SYSIN    DD *
           TEMPLATE A1 DSN(&DB..&SP..COPY1) UNIT CART STACK YES
           TEMPLATE A2 DSN(&DB..&SP..COPY2) UNIT CART STACK YES
COPY PARALLEL 2 TAPEUNITS 2
          TABLESPACE DSN8D81A.DSN8S81D COPYDDN(A1)
          INDEXSPACE DSN8810.XDEPT COPYDDN(A1)
          TABLESPACE DSN8D81A.DSN8S81E COPYDDN(A2)
          INDEXSPACE DSN8810.YDEPT COPYDDN(A2)
```

Although use of templates is recommended, you can also define the output data sets by coding JCL DD statements, as in Figure 22 on page 136. This COPY control

statement also specifies a list of objects to be processed in parallel, but in this case, the data sets are defined by DD statements. In each DD statement, notice the parameters for the VOLUME option. These values show that the data sets are defined on three different tape devices as follows:

- The first tape device contains data sets that are defined by DD statements DD1 and DD4. (For DD4, the VOLUME option has a value of *.DD1 for the REF parameter.)
- A second tape device contains data sets that are defined by DD statements DD2 and DD3. (For DD3, the VOLUME option has a value of *.DD3 for the REF parameter.)
- A third tape device contains the data set that is defined by DD statement DD5.

The following table spaces are to be processed in parallel on two different tape devices:

- DSN8D81A.DSN8S81D on the device that is defined by the DD1 DD statement and the device that is defined by the DD5 DD statement
- DSN8D81A.DSN8S81E on the device that is defined by the DD2 DD statement

Copying of the following tables spaces must wait until processing has completed for DSN8D81A.DSN8S81D and DSN8D81A.DSN8S81E:

- DSN8D81A.DSN8S81F on the device that is defined by the DD2 DD statement after DSN8D81A.DSN8S81E completes processing
- DSN8D81A.DSN8S81G on the device that is defined by the DD1 DD statement after DSN8D81A.DSN8S81D completes processing

```
//COPY1A   EXEC DSNUPROC,SYSTEM=DSN
//DD1 DD DSN=DB1.TS1.CLP,
//       DISP=(NEW,CATLG,CATLG),
//       UNIT=TAPE,LABEL=(1,SL),
//       VOLUME=(,RETAIN)
//DD2 DD DSN=DB2.TS2.CLP,
//       DISP=(NEW,CATLG,CATLG),
//       UNIT=TAPE,LABEL=(1,SL),
//       VOLUME=(,RETAIN)
//DD3 DD DSN=DB3.TS3.CLB.BACKUP,
//       DISP=(NEW,CATLG,CATLG),
//       UNIT=TAPE,LABEL=(2,SL),
//       VOLUME=(,RETAIN,REF=*.DD2)
//DD4 DD DSN=DB4.TS4.CLB.BACKUP,
//       DISP=(NEW,CATLG,CATLG),
//       UNIT=TAPE,LABEL=(2,SL),
//       VOLUME=(,RETAIN,REF=*.DD1)
//DD5 DD DSN=DB1.TS1.CLB.BACKUP,
//       DISP=(NEW,CATLG,CATLG),
//       UNIT=TAPE,LABEL=(1,SL),
//       VOLUME=(,RETAIN)
 COPY PARALLEL 2 TAPEUNITS 3
      TABLESPACE DSN8D81A.DSN8S81D COPYDDN(DD1,DD5)
      TABLESPACE DSN8D81A.DSN8S81E COPYDDN(DD2)
      TABLESPACE DSN8D81A.DSN8S81F COPYDDN(DD3)
      TABLESPACE DSN8D81A.DSN8S81G COPYDDN(DD4)
```

Figure 22. Example of making full image copies of a list of objects in parallel on tape

Example 5: Using both JCL-defined and template-defined data sets to copy a list of objects on tape: The example in Figure 23 on page 137 uses both JCL DD statements and utility templates to define four data sets for the image copies. The JCL defines two data sets (DB1.TS1.CLP and DB2.TS2.CLB.BACKUP), and the TEMPLATE utility control statements define two data sets that are to be

dynamically allocated (&DB..&SP..COPY1 and &DB..&SP..COPY2). For more information about TEMPLATE control statements, see “Syntax and options of the TEMPLATE control statement ” on page 593 in the TEMPLATE chapter.

The COPYDDN options in the COPY control statement specify the data sets that are to be used for the local primary and backup image copies of the specified table spaces. For example, the primary copy of table space DSN8D81A.DSN8S71D is to be written to the data set that is defined by the DD1 DD statement (DB1.TS1.CLP), and the primary copy of table space DSN8D81A.DSN8S71E is to be written to the data set that is defined by the A1 template (&DB..&SP..COPY1).

Four tape devices are allocated for this COPY job: the JCL allocates two tape drives, and the TAPEUNITS 2 option in the COPY statement indicates that two tape devices are to be dynamically allocated. Note that the TAPEUNITS option applies only to those tape devices that are dynamically allocated by the TEMPLATE statement.

Recommendation: Although this example shows how to use both templates and DD statements, use only templates, if possible.

```
//COPY1D   EXEC DSNUPROC,SYSTEM=DSN
//DD1 DD DSN=DB1.TS1.CLP,
//       DISP=(,CATLG),
//       UNIT=3490,LABEL=(1,SL)
//       VOLUME=(,RETAIN)
//DD2 DD DSN=DB2.TS2.CLB.BACKUP,
//       DISP=(,CATLG),
//       UNIT=3490,LABEL=(2,SL)
//       VOLUME=(,RETAIN)
//SYSIN    DD *
  TEMPLATE A1 DSN(&DB..&SN..COPY1) UNIT CART STACK YES
  TEMPLATE A2 DSN(&DB..&SN..COPY2) UNIT CART STACK YES
  COPY PARALLEL 2 TAPEUNITS 2
       TABLESPACE DSN8D81A.DSN8S81D COPYDDN(DD1,DD2)
       TABLESPACE DSN8D81A.DSN8S81E COPYDDN(A1,A2)
```

Figure 23. Example of using both JCL-defined and template-defined data sets to copy a list of objects on a tape

Example 6: Using LISTDEF to define a list of objects to copy in parallel to tape. The following example uses the LISTDEF utility to define a list of objects to be copied in parallel to different tape sources. The COPY control statement specifies that the table spaces that are included in the PAYROLL list are to copied. (The PAYROLL list is defined by the LISTDEF control statement.) The TEMPLATE control statements define two output data sets, one for the local primary copy (&DB..&COPY..LOCAL) and one for the recovery primary copy (&DB..&COPY..REMOTE).

```
//COPY3A   EXEC DSNUPROC,SYSTEM=DSN
//SYSIN    DD *
           LISTDEF PAYROLL INCLUDE TABLESPACES TABLESPACE DBPAYROLL.*
           TEMPLATE LOCAL DSN(&DB..&COPY..LOCAL) (+1) UNIT CART STACK YES
           TEMPLATE REMOTE DSN(&DB..&COPY..REMOTE) (+1) UNIT CART STACK YES
           COPY LIST PAYROLL PARALLEL(10) TAPEUNITS(8)
             COPYDDN(LOCAL) RECOVERYDDN(REMOTE)
```

In the preceding example, the utility determines the number of tape streams to use by dividing the value for TAPEUNITS (8) by the number of output data sets (2) for a total of 4 in this example. For each tape stream, the utility attaches one subtask. The list of objects is sorted by size and processed in descending order. The first

subtask to finish processes the next object in the list. In this example, the PARALLEL(10) option limits the number of objects to be processed in parallel to 10 and attaches four subtasks. Each subtask copies the objects in the list in parallel to two tape drives, one for the primary and one for the recovery output data sets.

For more information about LISTDEF control statements, see "Syntax and options of the LISTDEF control statement" on page 173 in the LISTDEF chapter. For more information about TEMPLATE control statements, see "Syntax and options of the TEMPLATE control statement " on page 593 in the TEMPLATE chapter.

Example 7: Making incremental copies with updates allowed. The FULL NO option in the following COPY control statement specifies that COPY is to make incremental image copies of any specified objects. In this case, the objects to be copied are those objects that are included in the NAME1 list, as indicated by the LIST option. The preceding LISTDEF utility control statement defines the NAME1 list to include index space DSN8D81A.XEMP1 and table space DSN8D81A.DSN8S81D. Although one of the objects to be copied is an index space and COPY does not take incremental image copies of index spaces, the job does not fail; COPY takes a full image copy of the index space instead. However, if a COPY FULL NO statement identifies only an index that is not part of a list, the COPY job fails.

All specified copies (local primary and backup copies and remote primary and backup copies) are written to data sets that are dynamically allocated according to the specifications of the COPYDS template. This template is defined in the preceding TEMPLATE utility control statement. For more information about templates, see Chapter 31, "TEMPLATE," on page 593.

The SHRLEVEL CHANGE option in the following COPY control statement specifies that updates can be made during the COPY job.

```
TEMPLATE COPYDS DSN &US.2.&SN..&LR.&PB..D&DATE.
  LISTDEF NAME1 INCLUDE INDEXSPACE DSN8D81A.XEMP1
                INCLUDE TABLESPACE DSN8D81A.DSN8S81D
  COPY LIST NAME1 COPYDDN(COPYDS, COPYDS) RECOVERYDDN(COPYDS,COPYDS)
  FULL NO SHRLEVEL CHANGE
```

Example 8: Making a conditional image copy. The CHANGELIMIT(5) option in the following control statement specifies the following conditions for making an image copy of table space DSN8D81P.DSN8S81C:

- Take a full image copy of the table space if the percentage of changed pages is equal to or greater than 5%.
- Take an incremental image copy of the table space if the percentage of changed pages is greater than 0 and less than 5%.
- Do not take an image copy if no pages have changed.

```
COPY TABLESPACE DSN8D81P.DSN8S81C CHANGELIMIT(5)
```

Example 9: Reporting image copy information for a table space. The REPORTONLY option in the following control statement specifies that image copy information is to be displayed only; no image copies are to be made. The CHANGELIMIT(10,40) option specifies that the following information is to be displayed:

- Recommendation that a full image copy be made if the percentage of changed pages is equal to or greater than 40%.
- Recommendation that an incremental image copy be made if the percentage of changed pages is greater than 10% and less than 40%.

- Recommendation that no image copy be made if the percentage of changed pages is 10% or less.

```
COPY TABLESPACE DSN8D81P.DSN8S81C CHANGELIMIT(10,40) REPORTONLY
```

Example 10: Invoking DFSMSdss concurrent copy. The CONCURRENT option in the following COPY control statement specifies that DFSMSdss concurrent copy is to make a full image copy of the objects in the COPYLIST list (table space DSN8D81A.DSN8S81D and table space DSN8D81A.DSN8S81P). The COPYDDN option indicates that the copy is to be written to the data set that is defined by the SYSCOPY1 template. The DSSPRINT DD statement specifies the data set for message output.

```
//COPY     EXEC DSNUPROC,SYSTEM=DSN
//SYSPRINT DD DSN=COPY1.PRINT1,DISP=(NEW,CATLG,CATLG),
//         SPACE=(4000,(20,20),,,ROUND),UNIT=SYSDA,VOL=SER=DB2CC5
//DSSPRINT DD DSN=COPY1.PRINT2,DISP=(NEW,CATLG,CATLG),
//         SPACE=(4000,(20,20),,,ROUND),UNIT=SYSDA,VOL=SER=DB2CC5
//SYSIN    DD *
  TEMPLATE SYSCOPY1 DSN &DB..&TS..COPY&IC.&LR.&PB..D&DATE..T&TIME.
           UNIT(SYSDA) DISP (MOD,CATLG,CATLG)
  LISTDEF COPYLIST INCLUDE TABLESPACE DSN8D81A.DSN8S81D
                   INCLUDE TABLESPACE DSN8D81A.DSN8S81P
  COPY LIST COPYLIST
  COPYDDN (SYSCOPY1)
  CONCURRENT
```

Figure 24. Example of invoking DFSMSdss concurrent copy with the COPY utility

Example 11: Invoking DFSMSdss concurrent copy and using a filter data set. The control statement in Figure 25 specifies that DFSMSdss concurrent copy is to make full image copies of the objects in the TSLIST list (table spaces TS1, TS2, and TS3). The FILTERDDN option specifies that COPY is to use the filter data set that is defined by the FILT template. All output is sent to the SYSCOPY data set, as indicated by the COPYDDN(SYSCOPY) option. SYSCOPY is the default. This data set is defined in the preceding TEMPLATE control statement.

```
LISTDEF TSLIST
        INCLUDE TABLESPACE TS1
        INCLUDE TABLESPACE TS2
        INCLUDE TABLESPACE TS3
TEMPLATE SYSCOPY DSN &DB..&TS..COPY&IC.&LR.&PB..D&DATE..T&TIME.
          UNIT(SYSDA) DISP (MOD,CATLG,CATLG)
TEMPLATE FILT DSN FILT.TEST1.&SN..D&DATE.
          UNIT(SYSDA) DISP (MOD,CATLG,DELETE)
COPY LIST TSLIST
FILTERDDN(FILT)
COPYDDN(SYSCOPY)
CONCURRENT
    SHRLEVEL REFERENCE
```

Figure 25. Example of invoking DFSMSdss concurrent copy with the COPY utility and using a filter data set

Example 12: Copying LOB table spaces together with related objects. Assume that table space TPIQUD01 is a base table space and that table spaces TLIQUDA1, TLIQUDA2, TLIQUDA3, and TLIQUDA4 are LOB table spaces. The control statement in Figure 26 on page 140 specifies that COPY is to take the following actions:

- Take a full image copy of each specified table space if the percentage of changed pages is equal to or greater than the highest decimal percentage value for the

CHANGELIMIT option for that table space. For example, if the percentage of changed pages for table space TPIQUD01 is equal to or greater than 6.7%, COPY is to take a full image copy.

- Take an incremental image copy of each specified table space if the percentage of changed pages falls in the range between the specified decimal percentage values for the CHANGELIMIT option for that table space. For example, if the percentage of changed pages for table space TLIQUDA1 is greater than 7.9% and less than 25.3%, COPY is to take an incremental image copy.
- Do not take an image copy of each specified table space if the percentage of changed pages is equal to or less than the lowest decimal percentage value for the CHANGELIMIT option for that table space. For example, if the percentage of changed pages for table space TLIQUDA2 is equal to or less than 2.2%, COPY is not to take an incremental image copy.
- Take full image copies of index spaces IPIQUD01, IXIQUD02, IUIQUD03, IXIQUDA1, IXIQUDA2, IXIQUDA3, and IXIQUDA4.

```
COPY
     TABLESPACE DBIQUD01.TPIQUD01 DSNUM ALL CHANGELIMIT(3.3,6.7)
       COPYDDN(COPYTB1)
     TABLESPACE DBIQUD01.TLIQUDA1 DSNUM ALL CHANGELIMIT(7.9,25.3)
       COPYDDN(COPYTA1)
     TABLESPACE DBIQUD01.TLIQUDA2 DSNUM ALL CHANGELIMIT(2.2,4.3)
       COPYDDN(COPYTA2)
     TABLESPACE DBIQUD01.TLIQUDA3 DSNUM ALL CHANGELIMIT(1.2,9.3)
       COPYDDN(COPYTA3)
     TABLESPACE DBIQUD01.TLIQUDA4 DSNUM ALL CHANGELIMIT(2.2,4.0)
       COPYDDN(COPYTA4)
     INDEXSPACE DBIQUD01.IPIQUD01 DSNUM ALL
       COPYDDN(COPYIX1)
     INDEXSPACE DBIQUD01.IXIQUD02 DSNUM ALL
       COPYDDN(COPYIX2)
     INDEXSPACE DBIQUD01.IUIQUD03 DSNUM ALL
       COPYDDN(COPYIX3)
     INDEXSPACE DBIQUD01.IXIQUDA1 DSNUM ALL
       COPYDDN(COPYIXA1)
     INDEXSPACE DBIQUD01.IXIQUDA2 DSNUM ALL
       COPYDDN(COPYIXA2)
     INDEXSPACE DBIQUD01.IXIQUDA3 DSNUM ALL
       COPYDDN(COPYIXA3)
     INDEXSPACE DBIQUD01.IXIQUDA4 DSNUM ALL
       COPYDDN(COPYIXA4)
 SHRLEVEL REFERENCE
```

Figure 26. Example of copying LOB table spaces together with related objects

Example 13: Using GDGs to make a full image copy. The following control statement specifies that the COPY utility is to make a full image copy of table space DBLT2501.TPLT2501. The local copies are to be written to data sets that are dynamically allocated according to the COPYTEM1 template. The remote copies are to be written to data sets that are dynamically allocated according to the COPYTEM2 template. For both of these templates, the DSN option indicates the name of generation data group JULTU225 and the generation number of +1. (If a GDG base does not already exist, DB2 creates one.) Both of these output data sets are to be modeled after the JULTU255.MODEL data set (as indicated by the MODELCB option in the TEMPLATE statements).

```
//*********************************************************
//* COMMENT: MAKE A FULL IMAGE COPY OF THE TABLESPACE.
//*          USE A TEMPLATE FOR THE GDG.
//*********************************************************
//STEP2    EXEC DSNUPROC,UID='JULTU225.COPY',
```

```
//         UTPROC='',
//         SYSTEM='SSTR'
//SYSIN    DD *
   TEMPLATE COPYTEM1
            UNIT SYSDA
            DSN 'JULTU225.GDG.LOCAL.&PB.(+1)'
            MODELDCB JULTU225.MODEL
   TEMPLATE COPYTEM2
            UNIT SYSDA
            DSN 'JULTU225.GDG.REMOTE.&PB.(+1)'
            MODELDCB JULTU225.MODEL
   COPY TABLESPACE DBLT2501.TPLT2501
        FULL YES
        COPYDDN (COPYTEM1,COPYTEM1)
        RECOVERYDDN (COPYTEM2,COPYTEM2)
        SHRLEVEL REFERENCE
```

Chapter 12. COPYTOCOPY

The COPYTOCOPY utility makes image copies from an image copy that was taken by the COPY utility. This includes inline copies that the REORG or LOAD utilities make. Starting with either the local primary or recovery-site primary copy, COPYTOCOPY can make up to three copies of one or more of the following types of copies:

- Local primary
- Local backup
- Recovery site primary
- Recovery site backup

You cannot run COPYTOCOPY on concurrent copies.

The RECOVER utility uses the copies when recovering a table space or index space to the most recent time or to a previous time. These copies can also be used by MERGECOPY, UNLOAD, and possibly a subsequent COPYTOCOPY execution.

For a diagram of COPYTOCOPY syntax and a description of available options, see "Syntax and options of the COPYTOCOPY control statement " on page 144. For detailed guidance on running this utility, see "Instructions for running COPYTOCOPY" on page 149.

Output: Output from the COPYTOCOPY utility consists of:

- Up to three sequential data sets that contain the image copy.
- Rows in the SYSIBM.SYSCOPY catalog table that describe the image copy data sets that are available to the RECOVER utility. Your installations responsible for ensuring that these data sets are available if the RECOVER utility requests them.

The entries for SYSCOPY columns remain the same as the original entries in the SYSCOPY row when the COPY utility recorded them. The COPYTOCOPY job inserts values in the columns DSNAME, GROUP_MEMBER, JOBNAME, AUTHID, DSVOLSER, and DEVTYPE.

Restrictions: COPYTOCOPY does not support the following catalog and directory objects:

- DSNDB01.SYSUTILX, and its indexes
- DSNDB01.DBD01, and its indexes
- DSNDB06.SYSCOPY, and its indexes

An image copy from a COPY job with the CONCURRENT option cannot be processed by COPYTOCOPY.

COPYTOCOPY does not check the recoverability of an object.

Related information: See Part 4 (Volume 1) of *DB2 Administration Guide* for uses of COPYTOCOPY in the context of planning for database recovery.

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:

- IMAGCOPY privilege for the database
- DBADM, DBCTRL, or DBMAINT authority for the database

- SYSCTRL or SYSADM authority

An ID with installation SYSOPR authority can also execute COPYTOCOPY, but only on a table space in the DSNDB01 or DSNDB06 database.

Execution phases of COPYTOCOPY: The COPYTOCOPY utility operates in these phases:

Phase	Description
UTILINIT	Performs initialization
CPY2CPY	Copies an image copy
UTILTERM	Performs cleanup

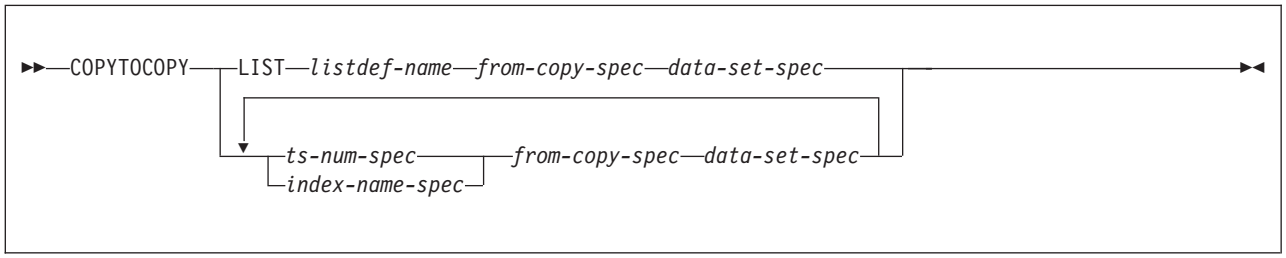
The following topics provide additional information:

- "Syntax and options of the COPYTOCOPY control statement "
- "Instructions for running COPYTOCOPY" on page 149
- "Concurrency and compatibility for COPYTOCOPY" on page 155
- "Sample COPYTOCOPY control statements" on page 156

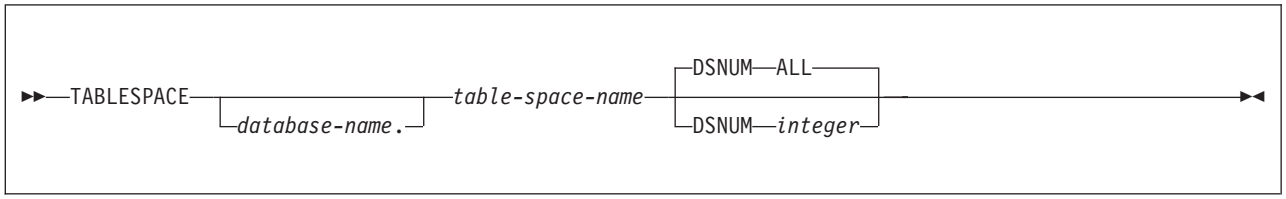
Syntax and options of the COPYTOCOPY control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

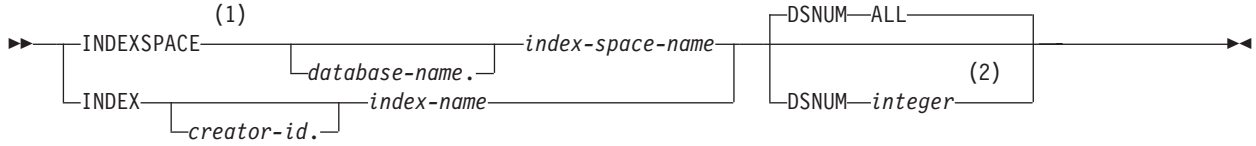
Syntax diagram



ts-num-spec:



index-name-spec:

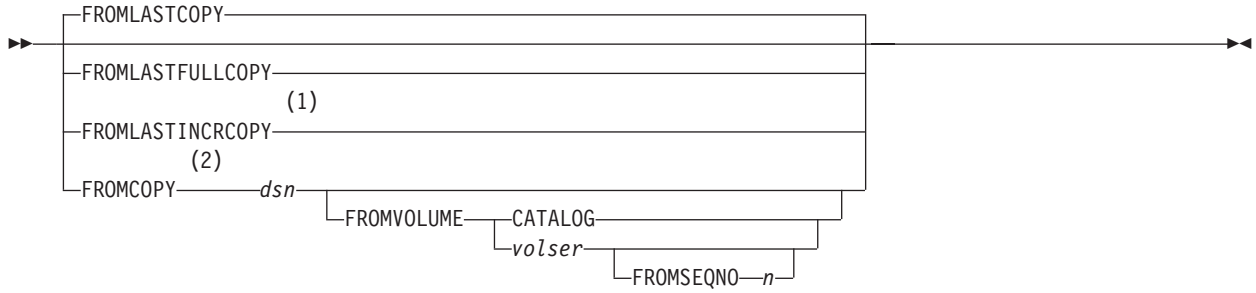


Notes:

1 INDEXSPACE is the preferred specification.

2 Not valid for nonpartitioning indexes.

from-copy-spec:

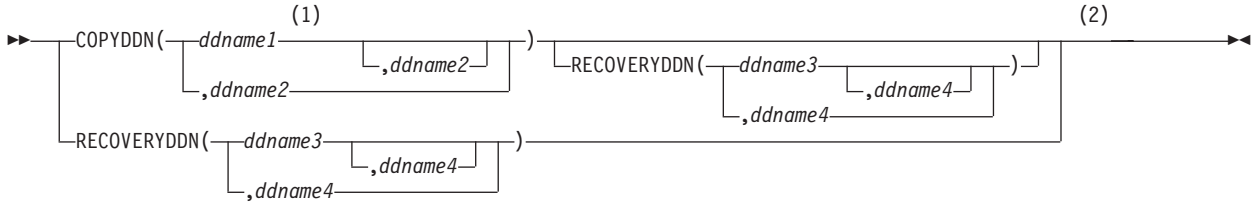


Notes:

1 Not valid with the INDEXSPACE or INDEX keyword.

2 Not valid with the LIST keyword.

data-set-spec:



Notes:

1 Use this option if you want to make a local site primary copy from one of the recovery site copies.

2 You can specify up to three DD names for both the COPYDDN and RECOVERYDDN options combined.

Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

LIST *listdef-name*
Specifies the name of a previously defined LISTDEF list name. The utility allows one LIST keyword for each COPYTOCOPY control statement. Do not specify LIST with either the INDEX or TABLESPACE keywords. DB2 invokes COPYTOCOPY once for the entire list. For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

TABLESPACE Specifies the table space (and, optionally, the database it belongs to) that is to be copied.

database-name is the name of the database that the table space belongs to. The **default** is **DSNDB04**.

table-space-name is the name of the table space to be copied.

INDEXSPACE *database-name.index-space-name*
Specifies the qualified name of the index space that is to be copied; the name is obtained from the SYSIBM.SYSINDEXES table. Define the index space with the COPY YES attribute.

database-name optionally specifies the name of the database that the index space belongs to. The **default** is **DSNDB04**.

index-space-name specifies the name of the index space that is to be copied.

INDEX *creator-id.index-name*
Specifies the index that is to be copied. Enclose the index name in quotation marks if the name contains a blank.

creator-id optionally specifies the creator of the index. The **default** is the user identifier for the utility.

index-name specifies the name of the index that is to be copied.

DSNUM Identifies a partition or data set, within the table space or the index

space, that is to be copied. The keyword ALL specifies that the entire table space or index space is to be copied.

ALL Specifies that the entire table space or index space is to be copied. The **default** is **ALL**. You must use ALL for a nonpartitioned secondary index.

integer Is the number of a partition or data set that is to be copied.

An integer value is not valid for nonpartitioned secondary indexes.

For a partitioned table space or index space, the integer is its partition number. The maximum is 4096.

For a nonpartitioned table space, find the integer at the end of the data set name as cataloged in the VSAM catalog. The data set name has the following format:

catname.DSNDB*x*.*dbname*.*spacename*.*y*0001.A*nnn*

In this format:

catname	Is the VSAM catalog name or alias.
x	Is C or D.
dbname	Is the database name.
spacename	Is the table space or index space name.
y	Is I or J.
nnn	Is the data set integer.

Specifying or using the default of DSNUM(ALL) causes COPYTOCOPY to look for an input image copy that was taken at the entire table space or index space level.

FROMLASTCOPY
Specifies the most recent image copy that was taken for the table space or index space that is to be the input to the COPYTOCOPY utility. This could be a full image copy or incremental copy that is retrieved from SYSIBM.SYSCOPY.

FROMLASTFULLCOPY
Specifies the most recent full image copy that was taken for the object, which is to be the input to the COPYTOCOPY job.

FROMLASTINCRCOPY
Specifies the most recent incremental image copy that was taken for the object that is to be the input to COPYTOCOPY job.

FROMLASTINCRCOPY is not valid with the INDEXSPACE or INDEX keyword. If FROMLASTINCRCOPY is specified for an INDEXSPACE or INDEX, COPYTOCOPY uses the last full copy that was taken, if one is available.

FROMCOPY *dsn*
Specifies a particular image copy data set (*dsn*) as the input to the COPYTOCOPY job. This option is not valid for LIST.

If the image copy data set is a generation data set, then supply a fully qualified data set name, including the absolute generation and version number. If the image copy data set is not a generation

data set and more than one image copy data set have the same data set name, use the FROMVOLUME option to identify the data set exactly.

FROMVOLUME
Identifies the image copy data set.

CATALOG
Identifies the data set as cataloged. Use this option only for an image copy that was created as a cataloged data set. (Its volume serial is not recorded in SYSIBM.SYSCOPY.)

COPYTOCOPY refers to the SYSIBM.SYSCOPY catalog table during execution. If you use FROMVOLUME CATALOG, the data set must be cataloged. If you remove the data set from the catalog after creating it, you must catalog the data set again to make it consistent with the record that appears in SYSIBM.SYSCOPY for this copy.

vol-ser
Identifies the data set by an alphanumeric volume serial identifier of its first volume. Use this option only for an image copy that was created as a noncataloged data set. Specify the first *vol-ser* in the SYSCOPY record to locate a data set that is stored on multiple tape volumes.If an individual volume serial number contains leading zeros, it must be enclosed in single quotation marks.

FROMSEQNO *n*
Identifies the image copy data set by its file sequence number. *n* is the file sequence number.

COPYDDN (*ddname1,ddname2***)**
Specifies a DD name (*ddname*) or a TEMPLATE name for the primary (*ddname1*) and backup (*ddname2*) copied data sets for the image copy at the local site. If *ddname2* is specified by itself, COPYTOCOPY expects the local site primary image copy to exist. If it does not exist, error message DSNU1401 is issued and the process for the object is terminated.

Recommendation: Catalog all of your image copy data sets.

You cannot have duplicate image copy data sets. If the DD statement identifies a noncataloged data set with the same name, volume serial, and file sequence number as one that is already recorded in SYSIBM.SYSCOPY, COPYTOCOPY issues a message and no copy is made. If the DD statement identifies a cataloged data set with only the same name, no copy is made. For cataloged image copy data sets, you must specify CATLG for the normal termination disposition in the DD statement; for example, DISP=(MOD,CATLG,CATLG). The DSVOLSER field of the SYSCOPY entry is blank.

When the image copy data set is going to a tape volume, specify VOL=SER parameter in the DD statement.

The COPYDDN keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name,

the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

RECOVERYDDN (*ddname3,ddname4***)**

Specifies a DD name (*ddname*) or a TEMPLATE name for the primary (*ddname3*) and backup (*ddname4*) copied data sets for the image copy at the recovery site. If *ddname4* is specified by itself, COPYTOCOPY expects the recovery site primary image copy to exist. If this image copy does not exist, error message DSNU1401 is issued and the process for the object is terminated.

You cannot have duplicate image copy data sets. The same rules apply for RECOVERYDDN as for COPYDDN.

The RECOVERYDDN keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

Instructions for running COPYTOCOPY

To run COPYTOCOPY, you must:

1. Read "Before running COPYTOCOPY" in this section.
2. Prepare the necessary data sets, as described in "Data sets that COPYTOCOPY uses."
3. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL and control statements for COPYTOCOPY, see "Sample COPYTOCOPY control statements" on page 156.)
4. Prepare a utility control statement, specifying the options for the tasks that you want to perform, as described in "Instructions for specific tasks" on page 151.
5. Plan for restart if the COPYTOCOPY job does not complete, as described in "Terminating or restarting COPYTOCOPY" on page 154.
6. Run COPYTOCOPY by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Before running COPYTOCOPY

Check the compatibility table in "Concurrency and compatibility for COPYTOCOPY" on page 155 if you want to run other jobs concurrently on the same target objects.

Data sets that COPYTOCOPY uses

Table 18 on page 150 describes the data sets that COPYTOCOPY uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 18. Data sets that COPYTOCOPY uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
Output copies	From one to three output data sets that contain the resulting image copy data sets. Specify their DD names with the COPYDDN and RECOVERYDDN options of the utility control statement.	Yes

The following objects are named in the utility control statement and do not require DD statements in the JCL:

Table space or Index space
Object that is to be copied. (If you want to copy only certain partitions in a partitioned table space, use the DSNUM option in the control statement.)

DB2 catalog objects
Objects in the catalog that COPYTOCOPY accesses. The utility records each copy in the DB2 catalog table SYSIBM.SYSCOPY.

Input image copy data set
This information is accessed through the DB2 catalog. However, if you want to preallocate your image copy data sets by using DD statements, see "Retaining tape mounts" on page 153 for more information. COPYTOCOPY retains all tape mounts for you.

Output data set size: Image copies are written to sequential non-VSAM data sets.

Recommendation: Use a template for the image copy data set for a table space by specifying a TEMPLATE statement without the SPACE keyword. When you omit this keyword, the utility calculates the appropriate size of the data set for you.

Alternatively, you can find the approximate size, in bytes, of the image copy data set for a table space by using the following procedure:

1. Find the *high-allocated page number* from the COPYPAGESF column of SYSIBM.SYSCOPY or from information in the VSAM catalog data set.
2. Multiply the high-allocated page number by the page size.

Another option is to look at the size of the input image copy.

JCL parameters: You can specify a block size for the output by using the BLKSIZE parameter on the DD statement for the output data set. Valid block sizes are multiples of 4096 bytes.

Cataloging image copies: To catalog your image copy data sets, use the DISP=(NEW,CATLG,CATLG) parameter in the DD statement or TEMPLATE that is named by the COPYDDN or RECOVERYDDN option. After the image copy is taken, the DSVOLSER column of the row that is inserted into SYSIBM.SYSCOPY contains blanks.

Duplicate image copy data sets are not allowed. If a cataloged data set is already recorded in SYSIBM.SYSCOPY with the same name as the new image copy data set, a message is issued and the copy is not made.

When RECOVER locates the entry in SYSIBM.SYSCOPY, it uses the ICF catalog to allocate the required data set. If you have uncataloged the data set, the allocation fails. In that case, the recovery can still go forward; RECOVER searches for a previous image copy. But even if RECOVER finds one, it must use correspondingly more of the log to recover. You are responsible for keeping the z/OS catalog consistent with SYSIBM.SYSCOPY with regard to existing image copy data sets.

Creating the control statement

Create the utility control statement for the COPYTOCOPY job. See “Syntax and options of the COPYTOCOPY control statement ” on page 144 for COPYTOCOPY syntax and option descriptions. See “Sample COPYTOCOPY control statements” on page 156 for examples of COPYTOCOPY usage.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in your utility control statement:

Copying full or incremental image copies

You can copy a full image copy or an incremental image copy by using FROMLASTCOPY keyword. If you do not specify FROMLASTCOPY, it will be used by default, as shown in the following example. In this example, the COPYTOCOPY control statement specifies that the utility is to make a backup copy of the most recent full image copy or an incremental image copy of the table space DSN8S81E in database DSN8D81A:

```
COPYTOCOPY TABLESPACE DSN8D81A.DSN8S81E
     COPYDDN(,DDNNAME2)
```

The COPYTOCOPY utility makes a copy from an existing image copy and writes pages from the image copy to the output data sets. The JCL for the utility job must include DD statements or a template for the output data sets. If the object consists of multiple data sets and all are copied in one job, the copies reside in one physical sequential output data set.

Copying incremental image copies

An incremental image copy is a copy of the pages that have changed since the last full or incremental image copy. To make a copy of an incremental image copy, use the keyword FROMLASTINCRCOPY.

The following example control statement specifies that COPYTOCOPY is to make a local site backup image copy, a recovery site primary image copy, and a recovery site backup image copy from an incremental image copy.

```
COPYTOCOPY TABLESPACE DSN8D81A.DSN8S81E
  FROMLASTINCRCOPY
  COPYDDN(,COPY2)
  RECOVERYDDN(COPY3,COPY4)
```

Using more than one COPYTOCOPY statement

You can use more than one control statement for COPYTOCOPY in one DB2 utility job step. After each COPYTOCOPY statement executes successfully:

- A row referring to the image copy is recorded in SYSIBM.SYSCOPY table.
- The image copy data set is valid and available for RECOVER, MERGECOPY, COPYTOCOPY, and UNLOAD.

If a job step that contains more than one COPYTOCOPY statement abnormally terminates, do not use TERM UTILITY. Restart the job from the last commit point by using RESTART instead. Terminating COPYTOCOPY in this case might cause inconsistencies between the ICF catalog and DB2 catalogs if generation data sets are used.

Copying an inline copy made by REORG with a range of partitions

COPYTOCOPY does not support a range of partitions within a partitioned table space. Specify individual DSNUM(*n*). From the inline copy, COPYTOCOPY copies only the specified partition into the output image copy data set.

Copying from a specific image copy

You can specify a particular image copy that is to be used as input to COPYTOCOPY by using FROMCOPY.The following control statement specifies that COPYTOCOPY is to make three copies of the table space TPA9031C in database DBA90301 from the image copy data set DH109003.COPY1.STEP1.COPY3:

```
COPYTOCOPY TABLESPACE DBA90301.TPA9031C
   FROMCOPY DH109003.COPY1.STEP1.COPY3
   COPYDDN(,COPY2)
   RECOVERYDDN(COPY3,COPY4)
```

If you specify the FROMCOPY keyword and the specified data set is not found in SYSIBM.SYSCOPY, COPYTOCOPY issues message DSNU1401I. Processing for the object then terminates.

Using TEMPLATE with COPYTOCOPY

Template data set name substitution variables resolve as usual. COPYTOCOPY does not use the template values of the original COPY utility execution.

Updating SYSCOPY records

The image copies COPYTOCOPY made are registered in SYSIBM.SYSCOPY for later use by the RECOVER utility. Other utilities can use these copies, too. Columns that are inserted by COPYTOCOPY are the same as those of the original entries in SYSCOPY row when the COPY utility recorded them. Except for columns GROUP_MEMBER, JOBNAME, AUTHID, DSNAME, DEVTYPE, and DSVOLSER, the columns are those of the COPYTOCOPY job. When COPYTOCOPY is invoked at the partition level (DSNUM *n*) and the input data set is an inline copy that was created by the REORG of a range of partitions, COPYTOCOPY inserts zeros in the HIGHDSNUM and LOWDSNUM columns of the SYSCOPY record.

Determining which input copy to use

If the FROMCOPY keyword is not specified, the COPYTOCOPY utility uses the following search order to determine the input data set for the utility:

- If you run the utility at the local site, the search order is the local site primary copy, the local site backup copy, the recovery site primary copy, and the recovery site backup copy.
- If you run the utility at the recovery site, the search order is the recovery site primary copy, the recovery site backup copy, the local site primary copy, and the local site backup copy.

If the input data set cannot be allocated or opened, COPYTOCOPY attempts to use the next image copy data set, with the same START_RBA value in SYSCOPY column, in the preceding search order.

If you use the FROMCOPY keyword, only the specified data set is used as the input to the COPYTOCOPY job.

Retaining tape mounts

COPYTOCOPY retains all tape mounts for you. You do not need to code JCL statements to retain tape mounts. If the image copy data sets that are used by COPYTOCOPY reside on the same tape, you do not need to remove the tape.

Defining generation data groups

Recommendation: Use generation data groups to hold image copies because their use automates the allocation of data set names and the deletion of the oldest data set.

Recommendation: Use templates when using generation data groups.

When you define the generation data group:

- You can specify that the oldest data set is to be automatically deleted when the maximum number of data sets is reached. If you do that, make the maximum number large enough to accommodate all recovery requirements. When data sets are deleted, use the MODIFY utility to delete the corresponding rows in SYSIBM.SYSCOPY.
- Make the limit number of generation data sets equal to the number of copies that you want to keep. Use NOEMPTY to avoid deleting all the data sets from the integrated catalog facility catalog when the limit is reached.

Using DB2 with DFSMS products

If image copy data sets are managed by HSM or SMS, all data sets are cataloged.

If you plan to use SMS, catalog all image copies. Never maintain cataloged and uncataloged image copies with the same name.

Putting image copies on tape

Do not combine a full image copy and incremental image copies for the same table space on one tape volume. If you do, the RECOVER TABLESPACE utility cannot allocate the incremental image copies.

Copying a LOB table space

You can make both full and incremental image copies of a LOB table space.

Copying a list of objects from tape

COPYTOCOPY determines the number of tape drives to use for the function. If you use JCL to define tape drives, the JCL allocates tape drives for those

definitions. If you use TEMPLATES to allocate tape drives for the output data sets, the utility dynamically allocates the tape drives according to the following algorithm:

- One tape drive if the input data set resides on tape.
- A tape drive for each template with STACK YES that references tape.
- Three tape drives, one for each of the local and remote output image copies, in case non-stacked templates reference tape.

Thus, COPYTOCOPY allocates a minimum of three tape drives. The utility allocates four tape drives if the input data set resides on tape, and more tape drives if you specified tape templates with STACK YES.

If input data sets to be copied are stacked on tape and output data sets are defined by a template, the utility sorts the list of objects by the file sequence numbers (FSN) of the input data sets and processes the objects serially.

For example, image copies of the following table spaces with their FSNs are stacked on **TAPE1:**

- DB2.TS1 FSN=1
- DB2.TS2 FSN=2
- DB2.TS3 FSN=3
- DB2.TS4 FSN=4

In the following statements, COPYTOCOPY uses a template for the output data set:

```
//COPYTOCOPY  EXEC DSNUPROC,SYSTEM=V71A
//SYSIN    DD *
TEMPLATE A1 &DB..&SP..COPY1 TAPE UNIT CART STACK YES
            COPYTOCOPY
            TABLESPACE DB1.TS4
                 LASTFULL
                 RECOVERYDDN(A1)
            TABLESPACE DB1.TS1
                 LASTFULL
                 RECOVERYDDN(A1)
            TABLESPACE DB1.TS2
                 LASTFULL
                 RECOVERYDDN(A1)
             TABLESPACE DB1.TS3
                 LASTFULL
                 RECOVERYDDN(A1)
```

As a result, the utility sorts the objects by FSN and processes them in the following order:

- DB1.TS1
- DB1.TS2
- DB1.TS3
- DB1.TS4

If the output data sets are defined by JCL, the utility gives stacking preference to the output data sets over input data sets. If the input data sets are not stacked, the utility sorts the objects by size in descending order.

Terminating or restarting COPYTOCOPY

This section explains how to terminate and restart the COPYTOCOPY utility.

Terminating COPYTOCOPY

You can use the TERM utility command to terminate a COPYTOCOPY job. For instructions on terminating an online utility, see "Terminating an online utility with the TERM UTILITY command" on page 42.

Restarting COPYTOCOPY

For instructions on restarting a utility job, see "Restarting an online utility" on page 43.

Restarting a COPYTOCOPY job: If you do **not** use the TERM UTILITY command, you can restart a COPYTOCOPY job. COPYTOCOPY jobs restart from the last commit point. You cannot use RESTART(PHASE) for any COPYTOCOPY job. If you are restarting a COPYTOCOPY job with uncataloged output data sets, you must specify the appropriate volumes for the job in the JCL or on the TEMPLATE utility statement. Doing so could impact your ability to use implicit restart.

To prepare for restarting a COPYTOCOPY job, specify DISP=(MOD,CATLG,CATLG) on your DD statements.

Restarting COPYTOCOPY after an out-of-space condition: See "Restarting after the output data set is full" on page 45 for guidance in restarting COPYTOCOPY from the last commit point after receiving an out-of-space condition.

Concurrency and compatibility for COPYTOCOPY

DB2 treats individual data and index partitions as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible.

Claims: Table 19 shows which claim classes COPYTOCOPY claims on the target object.

Table 19. Claim classes of COPYTOCOPY operations.

Target	COPYTOCOPY
Table space or partition, or index space or partition	UTRW

Legend:
- UTRW - Utility restrictive state - read-write access allowed

Compatibility: Table 20 documents which utilities can run concurrently with COPYTOCOPY on the same target object. The target object can be a table space, an index space, or a partition of a table space or index space. If compatibility depends on particular options of a utility, that information is also documented in the table.

Table 20. Compatibility of COPYTOCOPY with other utilities

Action	Compatible with COPYTOCOPY?
CHECK DATA	Yes
CHECK INDEX	Yes
CHECK LOB	Yes
COPY	No
DIAGNOSE	Yes

Table 20. Compatibility of COPYTOCOPY with other utilities (continued)

Action	Compatible with COPYTOCOPY?
LOAD	No
MERGECOPY	No
MODIFY	No
QUIESCE	Yes
REBUILD INDEX	Yes
RECOVER	No
REORG INDEX	No
REORG TABLESPACE	No
REPAIR	Yes
REPORT	Yes
RUNSTATS INDEX	Yes
RUNSTATS TABLESPACE	Yes
STOSPACE	Yes
UNLOAD	Yes

Sample COPYTOCOPY control statements

Example 1: Making a local backup copy. The following control statement specifies that the COPYTOCOPY utility is to make a local backup copy of the most recent full image copy or incremental image copy, whichever is most recent. The COPYDDN option specifies that the data set for the local site backup image copy is defined by the COPY2 DD statement. Because no data set is specified for the local site primary image copy, which is usually the first parameter of the COPYDDN option, COPYTOCOPY expects this copy to already exist. If it does not exist, DB2 issues an error message and terminates the job.

```
//STEP1    EXEC DSNUPROC,UID='DH109001.COPY1',
//         UTPROC='',
//         SYSTEM='DSN'
//COPY2    DD DSN=DH109001.C2C01.STEP2.COPY2,DISP=(MOD,CATLG,CATLG),
//         SPACE=(1000,(20,20),,,ROUND)
//SYSIN    DD *
   COPYTOCOPY TABLESPACE DBA90101.TLA9011A COPYDDN(,COPY2)
//
```

Example 2: Copying the most recent copy. The following control statement specifies that COPYTOCOPY is to make a local site backup copy, a recovery site primary copy, and a recovery site backup copy of table space DBA90102.TPA9012C. The COPYDDN and RECOVERYDDN options also indicate the data sets to which these copies should be written. For example, the recovery site primary copy is to be written to the COPY3 data set. The FROMLASTCOPY option specifies that the most recent full image copy or incremental image copy is to be used as the input copy data set. This option is the default and is therefore not required.

```
COPYTOCOPY TABLESPACE DBA90102.TPA9012C
      FROMLASTCOPY COPYDDN(,COPY2)
      RECOVERYDDN(COPY3,COPY4)
```

Example 3: Copying the most recent full image copy. The following control statement specifies that COPYTOCOPY is to make primary and backup copies at

the recovery site of table space DBA90201.TPA9021C. The FROMLASTFULLCOPY option specifies that the most recent full image copy is to be used as the input copy data set.

```
COPYTOCOPY TABLESPACE DBA90201.TPA9021C
      FROMLASTFULLCOPY
      RECOVERYDDN(COPY3,COPY4)
```

Example 4: Specifying a copy data set for input. The following control statement specifies that COPYTOCOPY is to make a local site backup copy, a recovery site primary copy, and a recovery site backup copy from data set DH109003.COPY1.STEP1.COPY3. This input data set is specified by the FROMCOPY option. The output data sets (COPY2, COPY3, and COPY4) are specified by the COPYDDN and RECOVERYDDN options.

```
COPYTOCOPY TABLESPACE DBA90301.TPA9031C
      FROMCOPY DH109003.COPY1.STEP1.COPY3
      COPYDDN(,COPY2)
      RECOVERYDDN(COPY3,COPY4)
```

Example 5: Identifying a cataloged image copy data set. The following control statement specifies that COPYTOCOPY is to make a local site backup copy from a cataloged data set that is named DH109003.COPY1.STEP1.COPY4. This data set is identified by the FROMCOPY and FROMVOLUME options. The FROMCOPY option specifies the input data set name, and the FROMVOLUME CATALOG option indicates that the input data set is cataloged. Use the FROMVOLUME option to distinguish a data set from other data sets that have the same name.

```
COPYTOCOPY TABLESPACE DBA90302.TLA9032A
      FROMCOPY DH109003.COPY1.STEP1.COPY4
      FROMVOLUME CATALOG
      COPYDDN(,COPY2)
```

Example 6: Identifying an uncataloged image copy data set. The COPYTOCOPY control statement in Figure 27 specifies that COPYTOCOPY is to make a local site backup copy, a recovery site primary copy, and a recovery site backup copy from an uncataloged data set, JUKQU2BP.COPY1.STEP1.TP01. The FROMCOPY option identifies this input data set name, and the FROMVOLUME option identifies the volume (SCR03) for the input data set. Use the FROMVOLUME option to distinguish a data set from other data sets that have the same name. The COPYDDN option identifies the data set for the local site backup copy. This data set is to be dynamically allocated according to the specifications of the C2C1_T1 template, which is defined in one of the preceding TEMPLATE control statements. The RECOVERYDDN option identifies the data sets for the recovery site copies. These data sets are to be dynamically allocated according to the specifications of the C2C1_T2 and C2C1_T3 templates, which are defined in the preceding TEMPLATE control statements. For more information about TEMPLATE control statements, see “Syntax and options of the TEMPLATE control statement ” on page 593 in the TEMPLATE chapter.

```
//STEP1    EXEC DSNUPROC,UID='JUKQU2BP.C2C1',
//         UTPROC='',
//         SYSTEM='SSTR'
//SYSIN    DD *

   TEMPLATE C2C1_T1
            DSN(JUKQU2BP.C2C1.LB.&SN.)
            DISP(NEW,CATLG,CATLG)
            UNIT(SYSDA)

   TEMPLATE C2C1_T2
            DSN(JUKQU2BP.C2C1.RP.&SN.)
            DISP(NEW,CATLG,CATLG)
            UNIT(SYSDA)

   TEMPLATE C2C1_T3
            DSN(JUKQU2BP.C2C1.RB.&SN.)
            DISP(NEW,CATLG,CATLG)
            UNIT(SYSDA)

   COPYTOCOPY TABLESPACE DBKQBP01.TPKQBP01
              FROMCOPY JUKQU2BP.COPY1.STEP1.TP01
              FROMVOLUME SCR03
              COPYDDN(,C2C1_T1)
              RECOVERYDDN(C2C1_T2,C2C1_T3)
/*
```

Figure 27. Example of identifying an uncataloged image copy data set

Example 7: Processing a list of objects. The following control statement specifies that COPYTOCOPY is to make local site backup copies of the three partitions of table space DBA90402.TPA9042C that are specified by the DSNUM option (partitions 2, 3, and 4). COPYTOCOPY uses the following input copy data sets, as indicated by the FROMLASTFULLCOPY, FROMLASTCOPY, and FROMLASTINCRCOPY options:

- The most recent full image copy for partition 2
- The most recent full image copy or incremental image copy, whichever is most recent, for partition 3
- The most recent incremental image copy for partition 4

The COPYDDN option for each partition indicates the output data sets (COPY2, COPY3, and COPY4).

```
COPYTOCOPY
      TABLESPACE DBA90402.TPA9042C DSNUM 2
          FROMLASTFULLCOPY COPYDDN(,COPY2)
      TABLESPACE DBA90402.TPA9042C DSNUM 3
          FROMLASTCOPY COPYDDN(,COPY3)
      TABLESPACE DBA90402.TPA9042C DSNUM 4
          FROMLASTINCRCOPY COPYDDN(,COPY4)
```

Example 8: Using LISTDEF and TEMPLATE. The following COPYTOCOPY control statement specifies that the utility is to copy the list of objects that are included in the CPY1 list, which is defined by the LISTDEF control statement. The copies are to be written to the data sets that are defined by the TMP1 template, which is defined in the TEMPLATE control statement. This template defines the naming convention for the output data sets that are to be dynamically allocated.

The OPTIONS PREVIEW statement before the LISTDEF statement is used to force the CPY1 list contents to be included in the output. For long lists, using this statement is not recommended, because it might cause the output to be too long.

The OPTIONS OFF statement ends the PREVIEW mode processing, so that the following TEMPLATE and COPYTOCOPY jobs run normally.

```
OPTIONS PREVIEW
   LISTDEF CPY1 INCLUDE TABLESPACES TABLESPACE DBA906*.T*A906*
           INCLUDE INDEXSPACES COPY YES INDEXSPACE ADMF001.I?A906*
   OPTIONS OFF
   TEMPLATE TMP1 UNIT SYSDA
                 DSN  (DH109006.COPY&LOCREM.&PRIBAC..&SN..T&TIME.)
                 DISP (MOD,CATLG,CATLG)
   COPYTOCOPY LIST CPY1 COPYDDN(TMP1,TMP1)
```

For more information about LISTDEF control statements, see “Syntax and options of the LISTDEF control statement” on page 173 in the LISTDEF chapter. For more information about TEMPLATE control statements, see “Syntax and options of the TEMPLATE control statement ” on page 593 in the TEMPLATE chapter. For more information about OPTIONS control statements, see “Syntax and options of the OPTIONS control statement” on page 317 in the OPTIONS chapter.

Chapter 13. DIAGNOSE

The DIAGNOSE online utility generates information that is useful in diagnosing problems. Use this utility only under the direction of IBM Software Support.

Interpreting output

One intended use of this utility is to aid in determining and correcting system problems. When diagnosing DB2 problems, you might need to refer to licensed documentation to interpret output from this utility.

For a diagram of DIAGNOSE syntax and a description of available options, see "Syntax and options of the DIAGNOSE control statement." For detailed guidance on running this utility, see "Instructions for running DIAGNOSE" on page 165.

Authorization required: To execute this utility for options which access relational data, you must use a privilege set that includes one of the following authorizations:

- REPAIR privilege for the database
- DBADM or DBCTRL authority for the database
- SYSCTRL or SYSADM authority

An ID with installation SYSOPR authority can execute the DIAGNOSE utility on a table space in the DSNDB01 or DSNDB06 database.

An ID with installation SYSADM authority can execute the DIAGNOSE utility with the WAIT statement option on any table space.

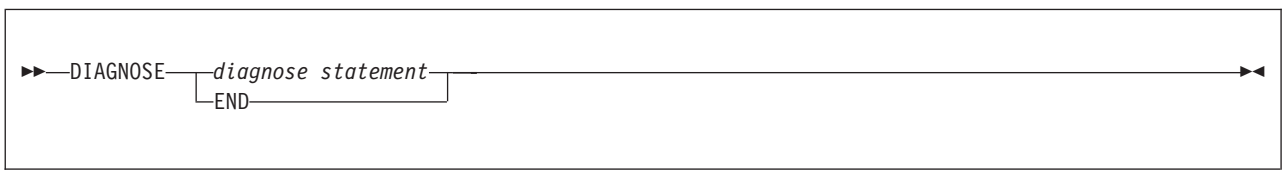
The following topics provide additional information:

- "Syntax and options of the DIAGNOSE control statement"
- "Instructions for running DIAGNOSE" on page 165
- "Concurrency and compatibility for DIAGNOSE" on page 166
- "Sample DIAGNOSE control statements" on page 166

Syntax and options of the DIAGNOSE control statement

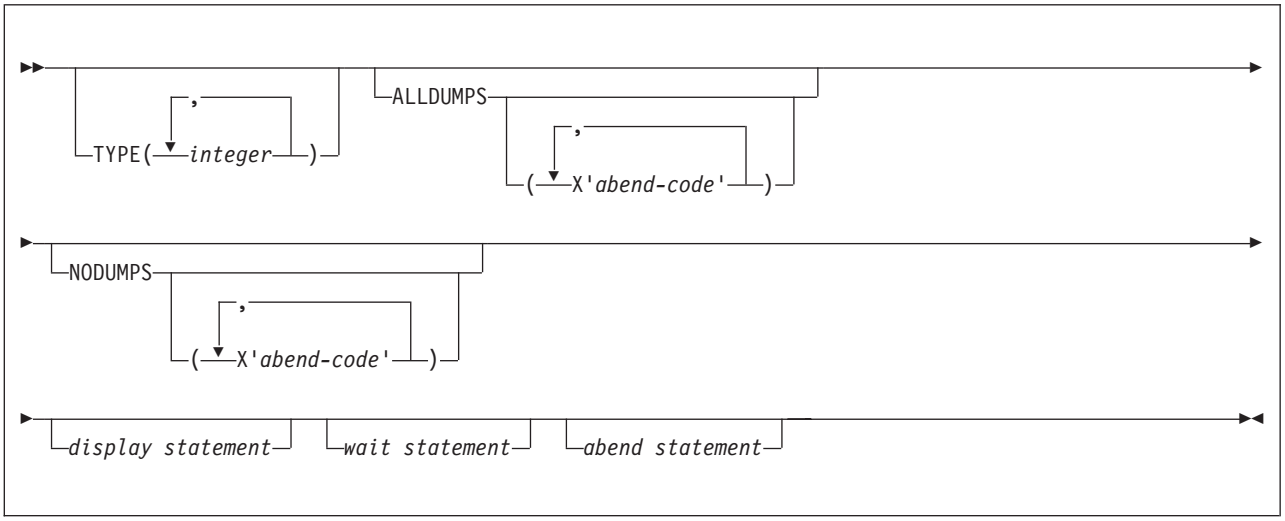
The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram

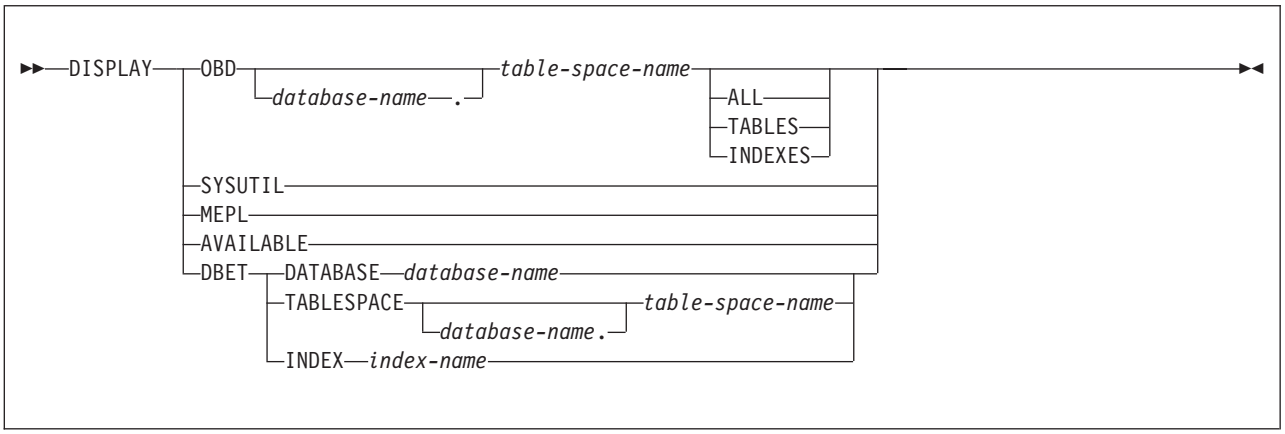


DIAGNOSE

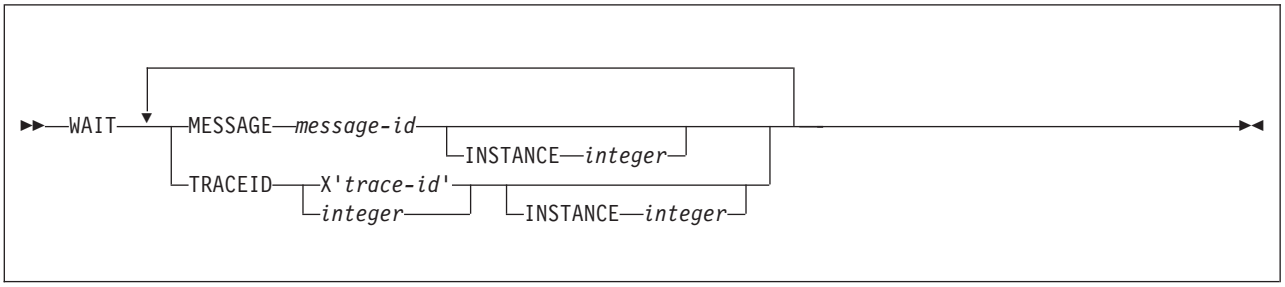
diagnose statement:



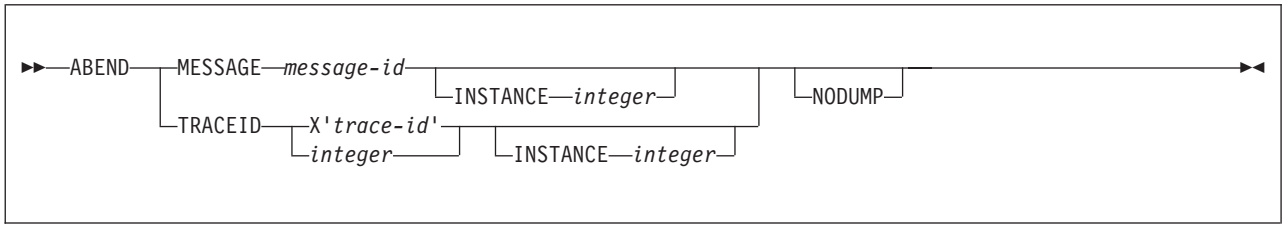
display statement:



wait statement:



abend statement:



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

TYPE(*integer***, ...)**
Specifies one or more types of diagnose that you want to perform.

integer is the number of types of diagnoses. The maximum number of types is 32. IBM Software Support defines the types as needed to diagnose problems with IBM utilities.

ALLDUMPS(X'*abend-code***', ...)**
Forces a dump to be taken in response to any utility abend code.

X'*abend-code*' is a member of a list of abend codes to which the scope of ALLDUMPS is limited.

abend-code is a hexadecimal value.

NODUMPS(X'*abend-code***', ...)**
Suppresses the dump for any utility abend code.

X'*abend-code*' is a member of a list of abend codes to which the scope of NODUMPS is limited.

abend-code is a hexadecimal value.

DISPLAY
Formats the specified database items using SYSPRINT.

OBD *database-name.table-space-name*
Formats the object descriptor (OBD) of the table space.

database-name is the name of the database in which the table space belongs.

table-space-name is the name of the table space whose OBD is to be formatted.

ALL Formats all OBDs of the table space. The OBD of any object that is associated with the table space is also formatted.

TABLES
Formats the OBDs of all tables in the specified table spaces.

INDEXES
Formats the OBDs of all indexes in the specified table spaces.

SYSUTIL
Formats every record from SYSIBM.SYSUTIL. This directory table stores information about all utility jobs.

MEPL
Dumps the module entry point lists (MEPLs) to SYSPRINT.

AVAILABLE
Displays the utilities that are installed on this subsystem in both bitmap and readable format. The presence or absence the utility products 5655-K61 (IBM DB2 Utilities Suite for z/OS) affects the results of this display. See message DSNU862I for the output of this display.

DBET
Dumps the contents of a database exception table (DBET) to SYSPRINT.

DATABASE *database-name*
Dumps the DBET entry that is associated with the specified database.

database-name is the name of the database.

TABLESPACE *database-name.table-space-name*
Dumps the DBET entry that is associated with the specified table space.

database-name is the name of the database.

table-space-name is the name of the table space.

INDEX *creator-name.index-name*
Dumps the DBET entry that is associated with the specified index.

creator-name is the ID of the creator of the index.

index-name is the name of the index.

Enclose the index name in quotation marks if the name contains a blank.

WAIT
Suspends utility execution when it encounters the specified utility message or utility trace ID. DIAGNOSE issues a message to the console and utility execution does not resume until the operator replies to that message, the utility job times out, or the utility job is canceled. This waiting period allows events to be synchronized while you are diagnosing concurrency problems. The utility waits for the operator to reply to the message, allowing the opportunity to time or synchronize events.

If neither the utility message nor the trace ID are encountered, processing continues.

ABEND
Forces an abend during utility execution if the specified utility message or utility trace ID is issued.

If neither the utility message nor the trace ID are encountered, processing continues.

NODUMP
Suppresses the dump that is generated by an abend of DIAGNOSE.

MESSAGE *message-id*
Specifies a DSNU*xxx* or DSNU*xxxx* message that causes a wait or an abend to occur when that message is issued. For information about the valid message IDs, see Part 2 of *DB2 Messages*.

message-id is the message, in the form of U*xxx* or U*xxxx*.

INSTANCE *integer*
Specifies that a wait or an abend is to occur when the MESSAGE option message has been encountered a specified number of times. If INSTANCE is not specified, a wait or abend occurs each time that the message is encountered.

integer is the number of times that a message is to be encountered before a wait or an abend occurs.

TRACEID *trace-id*
Specifies a trace ID that causes a wait or an abend to occur when the ID is encountered. You can find valid trace IDs can be found in data set *prefix*.SDSNSAMP(DSNWEIDS).

trace-id is a trace ID that is associated with the utility trace (RMID21). You can specify *trace-id* in either decimal (*integer*) or hexadecimal (X'*trace-id*') format.

INSTANCE *integer*
Specifies that a wait or an abend is to occur when the TRACEID option has been encountered a specified number of times. If INSTANCE is not specified, a wait or abend occurs each time that the trace ID is encountered.

integer is the number of times that a trace ID is to be encountered before a wait or an abend occurs.

END Ends DIAGNOSE processing.

Instructions for running DIAGNOSE

To run DIAGNOSE, you must:

1. Prepare the necessary data sets, as described in "Data sets that DIAGNOSE uses."
2. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for DIAGNOSE, see "Sample DIAGNOSE control statements" on page 166.)
3. Prepare a utility control statement that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks: Forcing a utility abend" on page 166.
4. Run DIAGNOSE by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Data sets that DIAGNOSE uses

Table 21 lists the data sets that DIAGNOSE uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set.

Table 21. Data sets that DIAGNOSE uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes

The following objects are named in the utility control statement and do not require DD statements in the JCL:

Database
Database about which DIAGNOSE is to gather diagnosis information.

Table space
Table space about which DIAGNOSE is to gather diagnosis information.

Index space
Index about which DIAGNOSE is to gather diagnosis information.

Instructions for specific tasks: Forcing a utility abend

To perform this task, specify the options and values for this task in your utility control statement.

DIAGNOSE can force a utility to abend when a specific message is issued. To force an abend when unique-index or referential-constraint violations are detected, you must specify the message that is issued when the error is encountered. Specify this message by using the MESSAGE option of the ABEND statement.

Instead of using a message, you can force an abend by using the TRACEID option of the ABEND statement to specify a trace IFCID that is associated with the utility to force an abend.

Use the INSTANCE keyword to specify the number of times that the specified message or trace record is to be generated before the utility abends.

Terminating or restarting DIAGNOSE

You can terminate a DIAGNOSE utility job by using the TERM UTILITY command if you submitted the job or have SYSOPR, SYSCTRL, or SYSADM authority.

You can restart a DIAGNOSE utility job, but it starts from the beginning again. For guidance in restarting online utilities, see “Restarting an online utility” on page 43.

Concurrency and compatibility for DIAGNOSE

DIAGNOSE can run concurrently on the same target object with any SQL operation or utility, except a utility that is running on DSNDB01.SYSUTILX.

Sample DIAGNOSE control statements

Example 1: Displaying DB2 MEPLs. The following DIAGNOSE utility control statement specifies that the DB2 MEPLs are to be displayed. You can use the output from this statement to find the service level of a specific DB2 module. The output lists each module, the most recent PTF or APAR that was applied to the module, and the date that the PTF or APAR was installed.

```
DIAGNOSE
 DISPLAY MEPL
```

Example 2: Forcing a dump. The following control statement forces a dump if an abend occurs with either of the following reason codes: X'00E40322' or X'00E40323'.

```
DIAGNOSE
 ALLDUMPS(X'00E40322',X'00E40323')
```

The following control statement forces a dump for any utility abend that occurs during the execution of the specified COPY job. The DIAGNOSE END option ends DIAGNOSE processing.

```
DIAGNOSE
 ALLDUMPS
 COPY TABLESPACE DSNDB06.SYSDBASE
DIAGNOSE END
```

Example 3: Performing a diagnosis of a specific type. The control statement in Figure 28 specifies that you want to perform a diagnosis of type 66. Run this job under the direction of IBM Software Support to diagnose problems with utility parallelism.

```
//STEP3    EXEC DSNUPROC,UID='JUOSU226.REBUI',
//            UTPROC='',SYSTEM='SSTR'
//SYSIN    DD *
DIAGNOSE TYPE(66)
    REBUILD INDEX (IDOS0302, IDOS0304, IPOS0301)
             SORTDEVT SYSDA SORTNUM 3
DIAGNOSE END
/*
```

Figure 28. Example of diagnosing type 66

Example 4: Forcing a utility abend. The control statement in Figure 29 forces an abend of the specified COPY job when one instance of message DSNU400 is issued. The NODUMP option indicates that the DIAGNOSE utility is not to generate a dump in this situation.

```
//STEP1    EXEC DSNUPROC,UID='IUJMU116.COPY1',
//         UTPROC='',
//         SYSTEM='DSN'
//SYSCOPY1  DD DSN=IUJMU116.COPY.STEP1.SYSCOPY1,DISP=(NEW,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
DIAGNOSE  ABEND MESSAGE U400
                INSTANCE 1
                NODUMP
                   COPY TABLESPACE DSN8D81A.DSN8S81E
                     COPYDDN SYSCOPY1
DIAGNOSE END
//*
```

Figure 29. Example of forcing an abend of the COPY utility

The following control statement forces an abend of the specified LOAD job when message DSNU311 is issued for the fifth time. The NODUMP option indicates that the DIAGNOSE utility is not to generate a dump in this situation.

```
DIAGNOSE
 ABEND MESSAGE U311 INSTANCE 5 NODUMP
LOAD DATA RESUME NO
  INTO TABLE TABLE1
   (NAME POSITION(1) CHAR(20))
DIAGNOSE END
```

Example 5: Displaying installed utilities. The following control statement specifies that a bitmap and list of all installed DB2 utilities are to be displayed.

```
DIAGNOSE DISPLAY AVAILABLE
```

The output from this utility job looks similar to the following output:

```
DSNU050I     DSNUGUTC - DIAGNOSE DISPLAY AVAILABLE
DSNU862I     DSNUDIAG - DISPLAY AVAILABLE UTILITIES.
                  MAP:  11111111111111111111110000000000000
-------------------------------------------------------------------------------
|CATMAINT |CHECK     |COPY      |DIAGNOSE |LISTDEF   |LOAD      |MERGECOPY|MODIFY    |
|OPTIONS  |QUIESCE   |REBUILD   |RECOVER  |REORG     |REPAIR    |REPORT   |RUNSTATS  |
|STOSPACE |TEMPLATE  |UNLOAD    |COPYTOCOP|EXEC      |BACKUP    |RESTORE  |          |
|         |          |          |         |          |          |         |          |
-------------------------------------------------------------------------------
```

Figure 30. Example output for DIAGNOSE DISPLAY AVAILABLE

The MAP output line shows a ″1″ for each installed utility. Each position represents a specific utility. This example output shows that the core utilities and the DB2 Utilities Suite are installed.

Example 6: Suspending utility execution. The control statement in Figure 31 indicates that the specified COPYTOCOPY job is to be suspended when it encounters 51 occurrences of the trace ID X'2E6F'.

```
//STEP2    EXEC DSNUPROC,UID='DH109012.C2C01',
//         UTPROC='',
//         SYSTEM='SSTR'
//COPY2    DD DSN=DH109012.C2C01.STEP2.COPY2,DISP=(MOD,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(1000,(20,20),,,ROUND)
//COPY3    DD DSN=DH109012.C2C01.STEP2.COPY3,DISP=(MOD,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(1000,(20,20),,,ROUND)
//COPY4    DD DSN=DH109012.C2C01.STEP2.COPY4,DISP=(MOD,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(1000,(20,20),,,ROUND)
//SYSIN    DD *
   DIAGNOSE WAIT TRACEID X'2E6F' INSTANCE 51
    COPYTOCOPY TABLESPACE DBA91201.TPA91201 DSNUM 1
               FROMLASTFULLCOPY COPYDDN(,COPY2)
                                RECOVERYDDN(COPY3,COPY4)
   DIAGNOSE END
/*
```

Figure 31. Example of suspending utility execution

Chapter 14. EXEC SQL

The EXEC SQL utility control statement declares cursors or executes dynamic SQL statements. You can use this utility as part of the DB2 cross-loader function of the LOAD utility. The cross-loader function enables you to use a single LOAD job to transfer data from one location to another location or from one table to another table at the same location. You can use either a local server or any DRDA-compliant remote server as a data input source for populating your tables. Your input can even come from other sources besides DB2 UDB for z/OS; you can use IBM Information Integrator Federation feature for access to data from sources as diverse as Oracle and Sybase, as well as the entire DB2 UDB family of database servers. For more information about using the cross loader function, see "Loading data by using the cross-loader function" on page 251.

Output: The EXEC SQL control statement produces a result table when you specify a cursor.

Authorization required: The EXEC SQL statement itself requires no privileges to execute. The authorization rules that are defined for the dynamic preparation of the SQL statement specified by EXECUTE IMMEDIATE apply. See *DB2 SQL Reference* for authorization rules for each SQL statement.

Execution phases of EXEC SQL: The EXEC SQL control statement executes entirely in the EXEC phase. You can restart the EXEC phase if necessary.

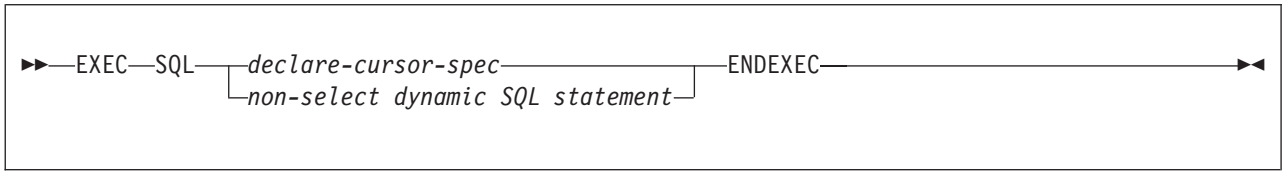
The following topics provide additional information:

- "Syntax and options of the EXEC SQL control statement"
- "Terminating or restarting EXEC SQL" on page 170
- "Concurrency and compatibility for EXEC SQL" on page 171
- "Sample EXEC SQL control statements" on page 171

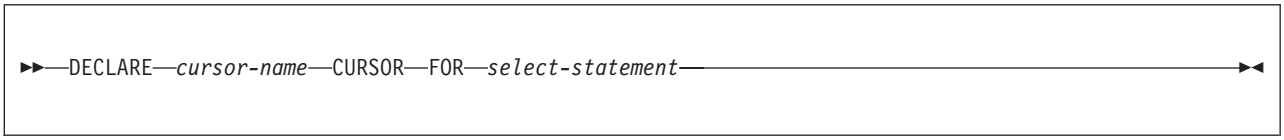
Syntax and options of the EXEC SQL control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram



declare-cursor-spec:



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

cursor-name Specifies the cursor name. The name must not identify a cursor that is already declared within the same input stream. When using the DB2 cross-loader function to load data from a remote server, you must identify the cursor with a three-part name. Cursor names that are specified with the EXEC SQL utility **cannot be longer than eight characters**.

select-statement Specifies the result table for the cursor. This statement can be any valid SQL SELECT statement, including joins, unions, conversions, aggregations, special registers, and user-defined functions. See *DB2 SQL Reference* for a description of the SELECT statement.

non-select dynamic SQL statement
Specifies a dynamic SQL statement that is to be used as input to EXECUTE IMMEDIATE. You can specify the following dynamic SQL statements in a utility statement:

ALTER	RENAME
COMMENT ON	REVOKE
COMMIT	ROLLBACK
CREATE	SET CURRENT DEGREE
DELETE	SET CURRENT LOCALE LC_CTYPE
DROP	SET CURRENT OPTIMIZATION HINT
EXPLAIN	SET PATH
GRANT	SET CURRENT PRECISION
INSERT	SET CURRENT RULES
LABEL ON	SET CURRENT SQLID
LOCK TABLE	UPDATE

Each SQL statement runs as a separate thread. When the utility executes the SQL statement, the specified statement string is parsed and checked for errors. If the SQL statement is invalid, EXEC SQL does not execute the statement and reports the error condition. If the SQL statement is valid, but an error occurs during execution, EXEC SQL reports that error condition. When an error occurs, the utility terminates.

Terminating or restarting EXEC SQL

You can terminate an EXEC SQL utility job by using the TERM UTILITY command if you submitted the job or have SYSOPR, SYSCTRL, or SYSADM authority.

You can restart an EXEC SQL utility job, but it starts from the beginning again. If you are restarting this utility as part of a larger job in which EXEC SQL completed successfully, but a later utility failed, do not change the EXEC SQL utility control statement, if possible. If you must change the EXEC SQL utility control statement,

use caution; any changes can cause the restart processing to fail. For guidance in restarting online utilities, see “Restarting an online utility” on page 43.

Concurrency and compatibility for EXEC SQL

You can use the EXEC SQL control statement with any utility that allows concurrent SQL access on a table space. Other databases are not affected.

Sample EXEC SQL control statements

Example 1: Creating a table: The following control statement specifies that DB2 is to create table MYEMP with the same rows and columns as sample table EMP.

```
EXEC SQL
  CREATE TABLE MYEMP LIKE DSN8810.EMP CCSID EBCDIC
ENDEXEC
```

This type of statement can be used to create a mapping table. For an example of creating and using a mapping table, see “Sample REORG TABLESPACE control statements” on page 486 in the REORG TABLESPACE chapter.

Example 2: Inserting rows into a table: The following control statement specifies that DB2 is to insert all rows from sample table EMP into table MYEMP.

```
EXEC SQL
  INSERT INTO MYEMP SELECT * FROM DSN8810.EMP
ENDEXEC
```

Example 3: Declaring a cursor: The following control statement declares C1 as the cursor for a query that is to return all rows from table DSN8810.EMP.

```
EXEC SQL
  DECLARE C1 CURSOR FOR SELECT * FROM DSN8810.EMP
ENDEXEC
```

You can use a declared cursor with the DB2 cross-loader function to load data from a local server or from any DRDA-compliant remote server as part of the DB2 cross-loader function. For more information about using the cross-loader function, see “Loading data by using the cross-loader function” on page 251.

Chapter 15. LISTDEF

The LISTDEF utility enables you to group database objects into reusable lists. You can then specify these lists in other utility control statements to indicate that the utility is to process all of the items in the list.

You can use LISTDEF to standardize object lists and the utility control statements that refer to them. This standardization reduces the need to customize or alter utility job streams.

If you do not use lists and you want to run a utility on multiple objects, you must run the utility multiple times or specify an itemized list of objects in the utility control statement.

Restriction: Objects that are created with the DEFINE NO attribute are excluded from all LISTDEF lists.

Output: Output from the LISTDEF control statement consists of a list with a name.

Authorization required: To execute the LISTDEF utility, you may require SELECT authority on SYSIBM.SYSINDEXES, SYSIBM.SYSTABLES, or SYSIBM.SYSTABLESPACE. Authority for each table is required only when that table is accessed during list expansion. Access varies depending on the keywords specified.

Additionally, you must have the authority to execute the utility that is used to process the list, as currently documented in the "Authorization required" section of each utility in this book.

Execution phases of LISTDEF: The LISTDEF control statement executes entirely within the UTILINIT phase.

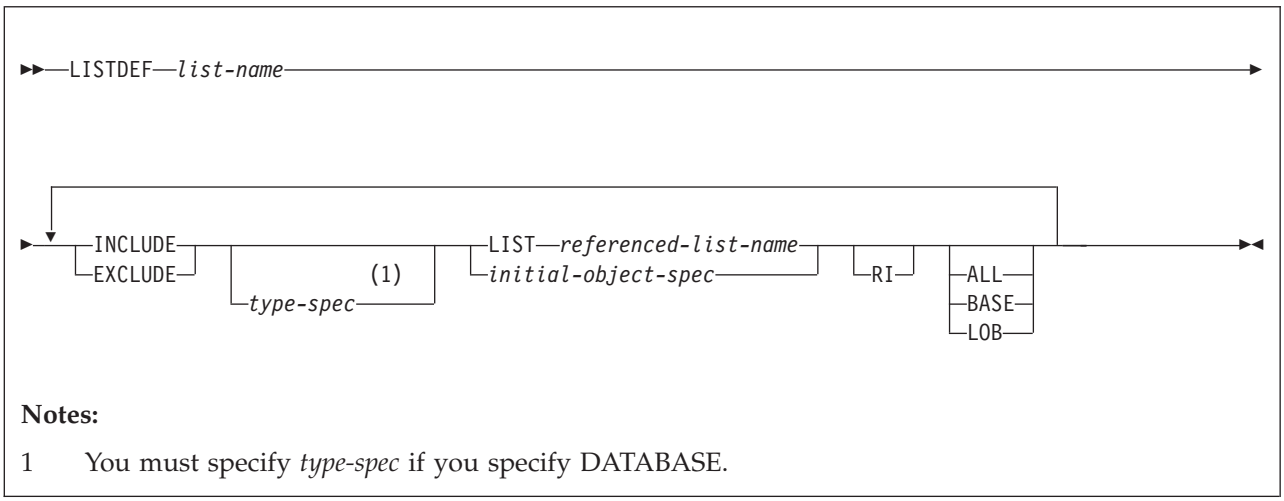
The following topics provide additional information:

- "Syntax and options of the LISTDEF control statement"
- "Instructions for using LISTDEF" on page 181
- "Concurrency and compatibility for LISTDEF" on page 187
- "Sample LISTDEF control statements" on page 188

Syntax and options of the LISTDEF control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

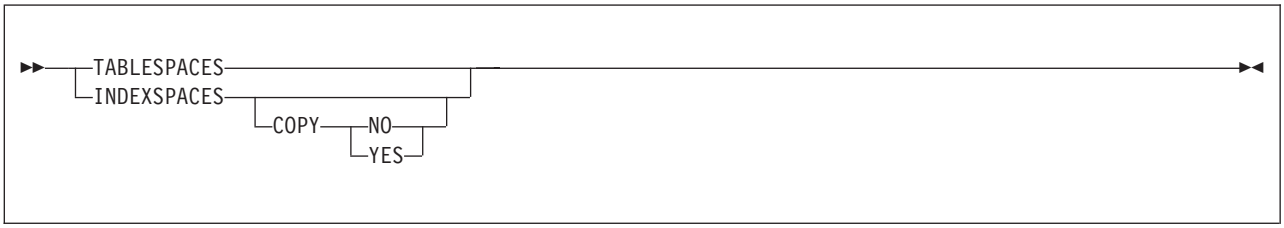
Syntax diagram



Notes:

1 You must specify *type-spec* if you specify DATABASE.

type-spec:



initial-object-spec:

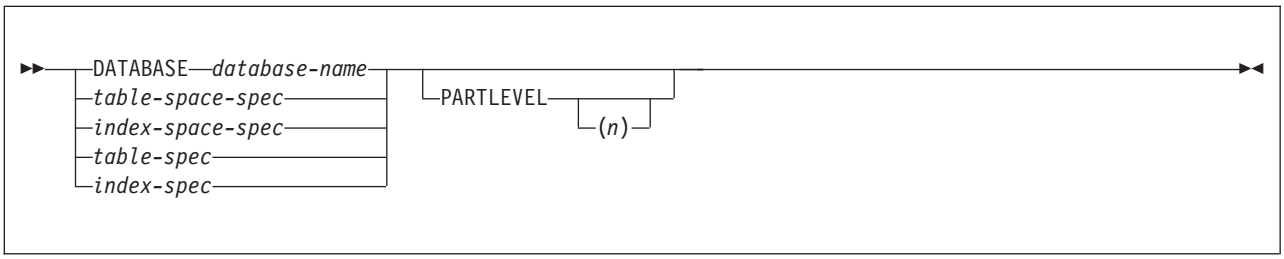
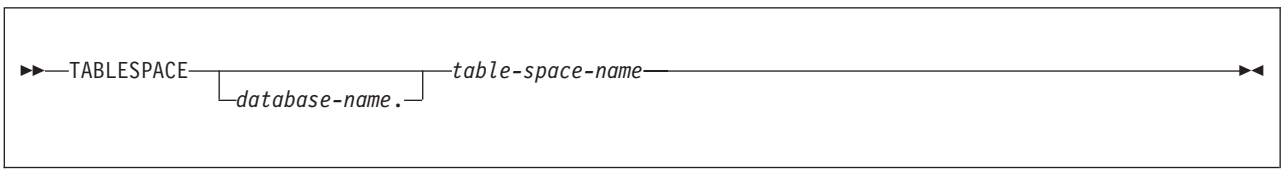


table-space-spec:



index-space-spec:

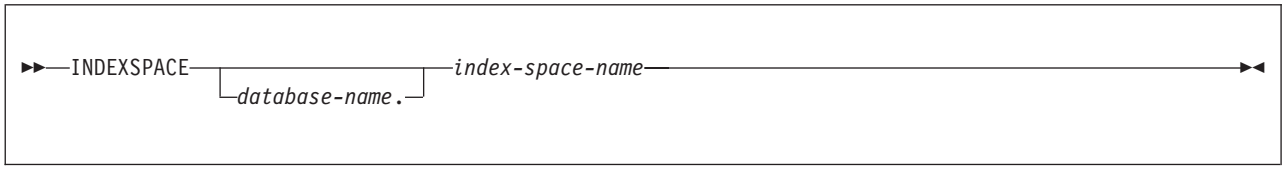
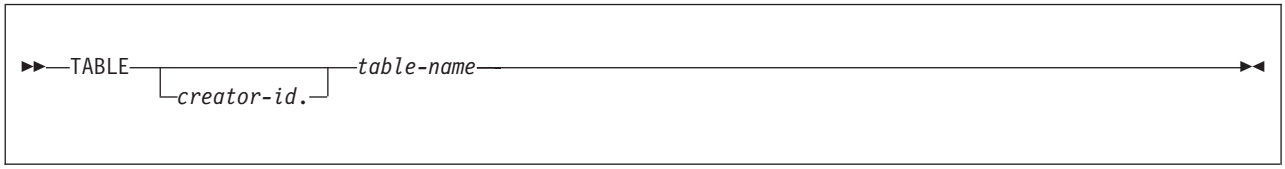
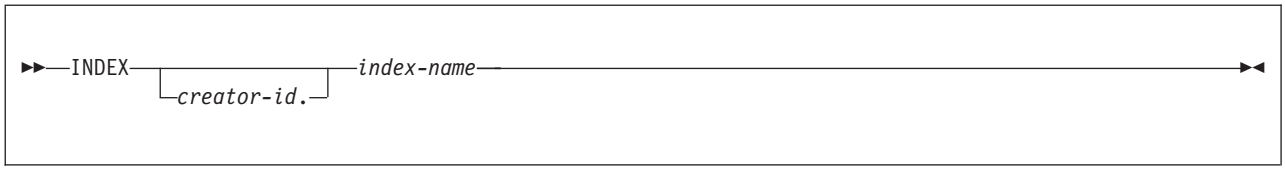


table-spec:



index-spec:



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

LISTDEF *list-name*

Defines a list of DB2 objects and assigns a name to the list. The list name makes the list available for subsequent execution as the object of a utility control statement or as an element of another LISTDEF statement.

list-name is the name (up to 18 alphanumeric characters in length) of the defined list.

You can put LISTDEF statements either in a separate LISTDEF library data set or before a DB2 utility control statement that refers to the *list-name*.

INCLUDE Specifies that the list of objects that results from the expression that follows is to be added to the list. You must first specify an INCLUDE clause. You can then specify subsequent INCLUDE or EXCLUDE clauses in any order to add to or delete clauses from the existing list. None of the objects referenced by this clause may be in the STOPPED state.

For detailed information about the order of INCLUDE and EXCLUDE processing, see "Including objects in a list" on page 181.

EXCLUDE Specifies, after the initial INCLUDE clause, that the list of objects that results from the expression that follows is to be excluded from the list if the objects are in the list. If the objects are not in the list,

they are ignored, and DB2 proceeds to the next INCLUDE or
EXCLUDE clause. None of the objects referenced by this clause
may be in the STOPPED state.

For detailed information about the order of INCLUDE and
EXCLUDE processing, see "Including objects in a list" on page 181.

TABLESPACES

Specifies that the INCLUDE or EXCLUDE object expression is to create a list of related table spaces.

TABLESPACES is the default type for lists that use a table space or a table for the initial search. For more information about specifying these objects, see the descriptions of the TABLESPACE and TABLE options.

No default type value exists for lists that use other lists for the initial search. The list that is referred to by the LIST option is used unless you specify TABLESPACES or INDEXSPACES. Likewise, no type default value exists for lists that use databases for the initial search. If you specify the DATABASE option, you must specify INDEXSPACES or TABLESPACES. For more information about specifying lists and databases, see the descriptions of the LIST and DATABASE options.

The result of the TABLESPACES keyword varies depending on the type of object that you specify in the INCLUDE or EXCLUDE clause. These results are shown in Table 22.

Table 22. Result of the TABLESPACES keyword based on the object type that is specified in the INCLUDE or EXCLUDE clause.

Object type specified in INCLUDE or EXCLUDE clause	Result of the TABLESPACES keyword
DATABASE	Returns all table spaces that are contained within the database
TABLESPACE	Returns the specified table space
TABLE	Returns the table space that contains the table
INDEXSPACE	Returns the table space that contains the related table
INDEX	Returns the table space that contains the related table
LIST of table spaces	Returns the table spaces from the expanded referenced list
LIST of index spaces	Returns the related table spaces for the index spaces in the expanded referenced list
LIST of table spaces and index spaces	Returns the table spaces from the expanded referenced list and the related table spaces for the index spaces in the same list

INDEXSPACES

Specifies that the INCLUDE or EXCLUDE object expression is to create a list of related index spaces.

INDEXSPACES is the default type for lists that use an index space or an index for the initial search. For more information about specifying these objects, see the descriptions of the INDEXSPACE and INDEX options.

No default type value exists for lists that use other lists for the initial search. The list that is referred to by the LIST option is used unless you specify TABLESPACES or INDEXSPACES. Likewise, no

type default value exists for lists that use databases for the initial search. If you specify the DATABASE option, you must specify INDEXSPACES or TABLESPACES. For more information about specifying lists and databases, see the descriptions of the LIST and DATABASE options.

The result of the INDEXSPACES keyword varies depending on the type of object that you specify in the INCLUDE or EXCLUDE clause. These results are shown in Table 23.

Table 23. Result of the INDEXSPACES keyword based on the object type that is specified in the INCLUDE or EXCLUDE clause.

Object type specified in INCLUDE or EXCLUDE clause	Result of the INDEXSPACES keyword
DATABASE	Returns all index spaces that are contained within the database
TABLESPACE	Returns all index spaces for indexes over all tables in the table space
TABLE	Returns all index spaces for indexes over the table
INDEXSPACE	Returns the specified index space.
INDEX	Returns the index space that contains the index
LIST of table spaces	Returns the related index spaces for the table spaces in the expanded referenced list
LIST of index spaces	Returns the index spaces from the expanded referenced list
LIST of table spaces and index spaces	Returns the index spaces from the expanded referenced list and the related index spaces for the table spaces in the same list

COPY Specifies whether indexes that were defined with or altered to COPY YES or COPY NO attributes are to be included or excluded in this portion of the list. If you omit COPY, all index spaces that satisfy the INCLUDE or EXCLUDE expression, regardless of their COPY attribute, are included or excluded in this portion of the list. If specified, this keyword must immediately follow the INDEXSPACES keyword. If you specify this keyword elsewhere, it is interpreted as the start of the COPY utility control statement.

YES Specifies that only index spaces that were defined with or altered to COPY YES are to be included in this portion of the list. Use INCLUDE with COPY YES to develop a list of index spaces that the COPY utility can process.

NO Specifies that only index spaces that were defined with or altered to COPY NO are to be included in this portion of the list. Use EXCLUDE with COPY NO to remove indexes that the COPY utility cannot process from a larger list.

LIST *referenced-list-name*

Specifies the name of a previously defined object list that is to be expanded and used for the initial search for the object.

referenced-list-name is the name of the list. You must explicitly specify this name. You cannot specify pattern-matching characters (%,*, ?, and _) for lists. None of the objects referenced by this clause may be in the STOPPED state.

No default type value exists for lists that are developed from the LIST option. The list is expanded as defined, and it is then modified by subsequent keywords, if any.

You can specify a *type-spec* of TABLESPACES to create a list of only table spaces. If the list to be processed contains index spaces, the TABLESPACES keyword creates a list that includes related table spaces.

You can specify a *type-spec* of INDEXSPACES to create a list of only index spaces. If the list to be processed contains table spaces, the INDEXSPACES keyword creates a list that includes related index spaces.

You can use the LIST keyword to make aggregate lists of lists, to exclude entire lists from other lists, and to develop lists of objects that are related to other lists.

DATABASE *database-name*
Specifies the database that is to be used for the initial search for the object.

You can specify the *database-name* explicitly or as a pattern-matched name. `DATABASE *` and `DATABASE %` are not supported.

If you specify DATABASE, you must also specify either TABLESPACES or INDEXSPACES as the list type. Depending on the list type that you specify, DB2 includes all table spaces or index spaces in *database-name* that satisfy the pattern-matching expression in the list.

You cannot specify DSNDB01, DSNDB06, DSNDB07, or user-defined work file databases in a LISTDEF.

TABLESPACE *database-name.table-space-name*
Specifies the table space that is to be used for the initial search for the object.

If you specify TABLESPACE, the default list type is
TABLESPACES. All table spaces that satisfy the pattern-matching
expression are included in the list unless the list is modified by
other keywords. `TABLESPACE *.*` and `TABLESPACE %.%` are not
supported.

database-name specifies the name of the database to which the table space belongs. The **default** is **DSNDB04**.

table-space-name specifies the name of the table space.

You can explicitly specify or use pattern-matching characters to specify *database-name*, *table-space-name*, or both.

You cannot include any objects in DSNDB07 or any user-defined work file databases in a LISTDEF. Pattern matching is not supported for DSNDB01 and DSNDB06 objects. None of the objects referenced by this clause may be in the STOPPED state.

INDEXSPACE *database-name.index-space-name*
Specifies the index space that is to be used for the initial search for the object.

If you specify INDEXSPACE, the default list type is
INDEXSPACES. All index spaces that satisfy the pattern-matching

expression are included in the list unless the index spaces are
excluded by other LISTDEF options. INDEXSPACE *.* and
INDEXSPACE %.% are not supported.

database-name specifies the name of the database to which the index space belongs. The **default** is **DSNDB04**.

index-space-name specifies the name of the index space.

You can explicitly specify or use pattern-matching characters to specify *database-name*, *index-space-name*, or both.

You cannot include any objects in DSNDB07 or any user-defined work file databases in a LISTDEF. Pattern-matching is not supported for DSNDB01 and DSNDB06 objects. None of the objects referenced by this clause may be in the STOPPED state.

TABLE *creator-id.table-name*

Specifies the table that is to be used for the initial search for the object.

If you specify TABLE, the default list type is TABLESPACES. All table spaces that contain tables that satisfy the pattern-matching expression are included in the list unless the list is modified by other keywords. TABLE *.* and TABLE %.% are not supported.

creator-id specifies the qualifier creator ID for the table. The **default** is the user identifier for the utility. *table-name* specifies the name of the table.

You can explicitly specify or use pattern-matching characters to specify *creator-id*,*table-name*, or both. However, the underscore pattern-matching character is ignored in a table name.

Pattern-matching is not supported for catalog and directory objects. In a LISTDEF statement, you must include catalog and directory objects by their fully qualified names. None of the objects referenced by this clause may be in the STOPPED state.

Enclose the table name in quotation marks if the name contains a blank.

INDEX *creator-id.index-name*

Specifies the index that is to be used for the initial search for the object.

If you specify INDEX, the default list type is INDEXSPACES. All index spaces that contain indexes that satisfy the pattern-matching expression are included in the list unless the list is modified by other keywords. INDEX *.* and INDEX %.% are not supported.

creator-id specifies the qualifier creator ID for the index. The **default** is the user identifier for the utility.

index-name specifies the name of the index.

Enclose the index name in quotation marks if the name contains a blank.

You can explicitly specify or use pattern-matching characters to specify *creator-id*, *index-name*, or both. However, the underscore pattern-matching character is ignored in an index name.

Pattern-matching is not supported for catalog and directory objects. In a LISTDEF statement, you must include catalog and directory objects by their fully qualified names. None of the objects referenced by this clause may be in the STOPPED state.

PARTLEVEL Specifies the partition granularity for partitioned table spaces, partitioning indexes, and data-partitioned secondary indexes that are to be contained in the list. You cannot specify the PARTLEVEL keyword with the RI keyword.

(*n***)** *n* is the integer partition number where *n* >= 0.

If you specify PARTLEVEL 0, the resulting list contains one entry for each nonpartitioned object.

If you specify PARTLEVEL with a nonzero operand, the resulting list contains one entry for the specified partition for partitioned objects and one entry for each nonpartitioned object.

If you specify PARTLEVEL without (*n*), the resulting list contains one entry for each partition in the partitioned object and one entry for each nonpartitioned object.

An INCLUDE with the PARTLEVEL keyword can be removed from the list only by an EXCLUDE with PARTLEVEL.

RI Specifies that all objects that are referentially related to the object expression (PRIMARY KEY <--> FOREIGN KEY) are to be included in the list. DB2 processes all referential relationships repeatedly until the entire referential set is developed. You cannot specify RI with PARTLEVEL(*n*).

LOB indicator keywords: Use one of three LOB indicator keywords to direct LISTDEF processing to follow auxiliary relationships to include related LOB objects in the list. The auxiliary relationship can be followed in either direction. LOB objects include the LOB table spaces, auxiliary tables, indexes on auxiliary tables, and their containing index spaces.

Incomplete LOB definitions cause seemingly related LOB objects to not to be found. The auxiliary relationship does not exist until you create the AUX TABLE with the STORES keyword.

No default LOB indicator keyword exists. If you do not specify BASE, LOB, or ALL, DB2 does not follow the auxiliary relationships and does not filter LOB from base objects in the enumerated list.

ALL
Specifies that both related BASE and LOB objects are to be included in the list. Auxiliary relationships are to be followed from all objects that result from the initial object lookup, and both BASE and LOB objects are to remain in the final enumerated list.

BASE
Specifies that only base table spaces (non-LOB) and index spaces are to be included in this element of the list.

If the result of the initial search for the object is a base object, auxiliary relationships are not followed. If the result of the initial search for the object is

a LOB object, the auxiliary relationship is applied to the base table space or index space, and only those objects become part of the resulting list.

LOB
Specifies that only LOB table spaces and related index spaces that contain indexes on auxiliary tables are to be included in this element of the list.

If the result of the initial search for the object is a LOB object, auxiliary relationships are not followed. If the result of the initial search for the object is a base object, the auxiliary relationship is applied to the LOB table space or index space, and only those objects become part of the resulting list.

Instructions for using LISTDEF

This section provides information about how to create and use object lists and includes the following topics:

Creating the control statement

The LISTDEF control statement defines a list of objects and assigns a name to the list. You must include the following elements in the control statement:

- The name of the list.
- An INCLUDE clause, optionally followed by additional INCLUDE or EXCLUDE clauses to either include or exclude objects from the list.

For a description of the elements that must be included in each INCLUDE and EXCLUDE clause, see "Specifying objects to include or exclude."

Including objects in a list

Use the INCLUDE and EXCLUDE clauses to specify the objects that are to be included in the list. Each INCLUDE clause adds objects to the list. Each EXCLUDE clause removes objects from the list. You must first specify an INCLUDE clause. You can then specify subsequent INCLUDE or EXCLUDE clauses in any order to add to or delete objects from the existing list.

DB2 constructs the list, one clause at a time, by adding objects to or removing objects from the list. If an EXCLUDE clause attempts to remove an object that is not yet in the list, DB2 ignores the EXCLUDE clause of that object and proceeds to the next INCLUDE or EXCLUDE clause. Be aware that a subsequent INCLUDE can return a previously excluded object to the list.

You must specify either INCLUDE or EXCLUDE. No default specification exists.

Specifying objects to include or exclude

Each INCLUDE or EXCLUDE clause identifies specific objects to add to or remove from the list.

You must include the following elements in each INCLUDE or EXCLUDE clause:

- The object that is to be used in the initial catalog lookup for each INCLUDE or EXCLUDE clause. The search for objects can begin with databases, table spaces, index spaces, tables, indexes, or other lists. You can explicitly specify the names of these objects or, with the exception of other lists, use a pattern matching expression. The resulting list contains only table spaces, only index spaces, or both.
- The type of objects that the list contains, either TABLESPACES or INDEXSPACES. You must explicitly specify the list type only when you specify a database as the initial object by using the keyword DATABASE. Otherwise, LISTDEF uses the default list type values shown in Table 24. These values depend on the type of object that you specified for the INCLUDE or EXCLUDE clause.

Table 24. Default list type values that LISTDEF uses.

Specified object	Default list type value
TABLESPACE	TABLESPACES
TABLE	TABLESPACES
INDEXSPACE	INDEXSPACES
INDEX	INDEXSPACES
LIST	Existing type value of the list

For example, the following INCLUDE clause specifies that table space DBLT0301.TLLT031A is to be added to the LIST:

```
INCLUDE TABLESPACE DBLT0301.TLLT031A
```

In the preceding example, table space DBLT0301.TLLT031A is specified as the object that LISTDEF is to use for the initial catalog lookup. By default, the list type value for a TABLESPACE object is TABLESPACES. Therefore, the list includes only table space DBLT0301.TLLT031A.

The following example INCLUDE clause is similar to the preceding example, except that it includes the INDEXSPACES keyword:

```
INCLUDE INDEXSPACES TABLESPACE DBLT0301.TLLT031A
```

In this example, the clause specifies that all index spaces over all tables in table space DBLT0301.TLLT031A are to be added to the list.

Optionally, you can add related objects to the list by specifying keywords that indicate a relationship, such as referentially related objects or auxiliary related objects. Valid specifications include the following keywords:

- BASE (non-LOB objects)
- LOB (LOB objects)
- ALL (both BASE and LOB objects)
- TABLESPACES (related table spaces)
- INDEXSPACES (related index spaces)
- RI (related by referential constraints, including informational referential constraints)

The preceding keywords perform two functions: they determine which objects are related, and they then filter the contents of the list. The behavior of these keywords varies depending on the type of object that you specify. For example, if your initial object is a LOB object, the LOB keyword is ignored. If, however, the initial object is not a LOB object, the LOB keyword determines which LOB objects are related, and

DB2 excludes non-LOB objects from the list. For more information about the keywords that can be used to indicate relationships, see“Option descriptions” on page 175.

DB2 processes each INCLUDE and EXCLUDE clause in the following order:

1. Perform the initial search for the object that is based on the specified pattern-matching expression, including PARTLEVEL specification, if specified.
2. Add or remove related objects and filter the list elements based on the specified list type, either TABLESPACES or INDEXSPACES (COPY YES or COPY NO).
3. Add or remove related objects depending on the presence or absence of the RI, BASE, LOB, and ALL keywords.

For example, to generate a list of all table spaces in the ACCOUNT database but exclude all LOB table spaces, you can specify the following LISTDEF statement:

```
LISTDEF ACCNT INCLUDE TABLESPACES DATABASE ACCOUNT BASE
```

In the preceding example, the name of the list is ACCNT. The TABLESPACES keyword indicates that the list is to include table spaces that are associated with the specified object. In this case, the table spaces to be included are those table spaces in database ACCOUNT. Finally, the BASE keyword limits the objects to only base table spaces.

If you want a list of only LOB index spaces in the ACCOUNT database, you can specify the following LISTDEF statement:

```
LISTDEF ACLOBIX INCLUDE INDEXSPACES DATABASE ACCOUNT LOB
```

In the preceding example, the INDEXSPACES and LOB keywords indicate that the INCLUDE clause is to add only LOB index spaces to the ACLOBIX list.

Restriction: Utilities do not support SYSUTILX-related objects inside a LISTDEF specification. You cannot specify the following objects in a LISTDEF:

- TABLESPACE DSNDB01.SYSUTILX
- TABLE SYSIBM.SYSUTILX
- TABLE SYSIBM.SYSUTIL
- INDEXSPACE DSNDB01.DSNLUX01
- INDEXSPACE DSNDB01.DSNLUX02
- INDEX SYSIBM.DSNLUX01
- INDEX SYSIBM.DSNLUX02

Using pattern matching expressions

You can use four special pattern-matching characters (%, *, _,?) to define generic object names in a LISTDEF statement. These characters are similar to those characters that are used in the SQL LIKE predicate. Utilities that reference a list access the DB2 catalog at execution time and dynamically expand each generic object name into an equivalent enumerated list. A utility processes this enumerated list either sequentially or in parallel, depending on the utility function and the parameters that you specify.

Restriction: DB2 does not support all-inclusive lists (such as `DATABASE *` or `TABLESPACE *.*`).

Restriction: Pattern-matching of DB2 catalog and directory objects (DSNDB06 and DSNDB01) is not supported. Catalog and directory objects must be included in a LISTDEF by their full table space or index space name. Even if catalog and

directory objects match a LISTDEF pattern matching expression, they are not included in the list. To process those objects, you must use syntax from releases prior to Version 7.

Specify pattern-matching object names by using the pattern-matching characters that are shown in Table 25. This table lists the pattern-matching character, the equivalent SQL symbol, and any additional information.

Table 25. LISTDEF pattern-matching characters

LISTDEF pattern-matching character	Equivalent symbol used in SQL LIKE predicates	Usage notes
Percent sign (%)	Percent sign (%)	Performs the same function.
Question mark (?)	Underscore (_)	Use the question mark (?) instead of underscore (_) as a pattern-matching character in table and index names. The underscore character (_) in table and index names represents a single occurrence of itself.
Asterisk (*)	Percent sign (%)	Performs the same function.
Underscore (_)	Underscore (_)	Use the underscore (_) as an alternative to the question mark (?) for database, table space, and index space names.

Including catalog and directory objects

If you specify DB2 directory objects (DSNDB01) and DB2 catalog objects (DSNDB06) in object lists, you must specify the fully qualified table space or index space names for those objects. Pattern-matching is not supported for catalog or directory objects. DB2 issues error messages for any catalog or directory objects that are invalid for a utility.

Although DB2 catalog and directory objects can appear in LISTDEF lists, these objects might be invalid for a utility and result in an error message.

The following valid INCLUDE clauses contain catalog and directory objects:

- `INCLUDE TABLESPACE DSNDB06.SYSDBASE`
- `INCLUDE TABLESPACES TABLESPACE DSNDB06.SYSDBASE`
- `INCLUDE INDEXSPACE DSNDB06.DSNDXX01`
- `INCLUDE INDEXSPACES INDEXSPACE DSNDB06.DSNDXX01`

Restriction: If you specify a catalog or directory object in a LISTDEF control statement, you cannot specify the following keywords:

- DATABASE
- TABLE
- INDEX
- BASE
- LOB
- ALL
- Databases DSNDB01, DSNDB06, and DSNDB07
- Table or indexes with a creator id of SYSIBM

These keywords require DB2 to access the catalog, which can cause problems when you specify a catalog or directory object.

All LISTDEF lists automatically exclude work file databases, which consist of DSNDB07 objects and user-defined work file objects, because DB2 utilities do not process these objects.

Previewing the contents of a list

You can preview the objects that are to be included in a list by using the PREVIEW function. When you run a utility using the PREVIEW function, DB2 expands any LISTDEF control statements into the equivalent enumerated list, prints it to SYSPRINT, and stops execution.

Specify PREVIEW in one of two ways, either as a JCL parameter or on the OPTIONS PREVIEW control statement. For details about the OPTIONS PREVIEW statement, see "Syntax and options of the LISTDEF control statement" on page 173.

Creating LISTDEF libraries

You can create a library of LISTDEF control statements by using a DD statement to name LISTDEF data sets.

For example, assume that data sets ADMF001.DB.LIST1 and ADMF001.DB.LIST2 each contain several LISTDEF statements. For any utility jobs that reference these LISTDEF statements, you can include the following DD statement in the JCL:

```
//LISTDSN  DD DSN=ADMF001.DB.LIST1,DISP=SHR
//         DD DSN=ADMF001.DB.LIST2,DISP=SHR
```

This DD statement defines a LISTDEF library. The statement gives a name (LISTDSN) to a group of data sets that contain LISTDEF statements, in this case ADMF001.DB.LIST1 and ADMF001.DB.LIST2. Defining such a library enables you to subsequently refer to the LISTDEF statements in that library by using the OPTIONS LISTDEFDD control statement.

Any data sets that are identified as part of a LISTDEF library must contain only LISTDEF statements.

In the utility job that references those LISTDEF statements, include an OPTIONS statement before the utility statement. In the OPTIONS statement, specify the DD name of the LISTDEF library as LISTDEFDD *ddname*.

DB2 uses this LISTDEF library for any subsequent utility control statements, until either the end of input or until you specify another OPTIONS LISTDEFDD *ddname*. The default DD name for the LISTDEF definition library is SYSLISTD.

When DB2 encounters a reference to a list, DB2 first searches SYSIN. If DB2 does not find the definition of the referenced list, DB2 searches the specified LISTDEF library.

Using lists in other utility jobs

This section explains how to reference lists in other utility jobs.

Placing the LISTDEF control statement

Specify LISTDEF control statements in the SYSIN DD statement prior to the utility control statement that references it, or in one or more LISTDEF library data sets. For more information about LISTDEF libraries and how to create them, see "Creating LISTDEF libraries."

Any LISTDEF statement that is defined within the SYSIN DD statement overrides another LISTDEF definition of the same name found in a LISTDEF library data set.

Referencing the list in another utility control statement

To use a list that has been defined with the LISTDEF control statement as the target object for a specific utility, specify the list name, prefixed by the LIST keyword. For example, you could QUIESCE all objects in a list by specifying the following control statement:

```
QUIESCE LIST X
```

In general, utilities processes the objects in the list in the order in which they are specified. However, some utilities alter the list order for optimal processing as follows:

- CHECK INDEX, REBUILD INDEX, and RUNSTATS INDEX process all index spaces that are related to a given table space at one time, regardless of list order.
- UNLOAD processes all specified partitions of a given table space at one time regardless of list order.

The LIST keyword is supported by the utilities that are listed in Table 26. When possible, utility processing optimizes the order of list processing as indicated in the table.

Table 26. How specific utilities process lists

Utility	Order of list processing
CHECK INDEX	Items are grouped by related table space.
COPY	Items are processed in the specified order on a single call to COPY; the PARALLEL keyword is supported.
COPYTOCOPY	Items are processed in the specified order on a single call to COPYTOCOPY.
MERGECOPY	Items are processed in the specified order.
MODIFY RECOVERY	Items are processed in the specified order.
MODIFY STATISTICS	Items are processed in the specified order.
QUIESCE	All items are processed in the specified order on a single call to QUIESCE.
REBUILD	Items are grouped by related table space.
RECOVER	Items are processed in the specified order on a single call to RECOVER.
REORG	Items are processed in the specified order.
REPORT	Items are processed in the specified order.
RUNSTATS INDEX	Items are grouped by related table space.
RUNSTATS TABLESPACE	Items are processed in the specified order.
UNLOAD	Items at the partition level are grouped by table space.

Some utilities such as COPY and RECOVER, can process a LIST without a specified object type. Object types are determined from the list contents. Other utilities, such as REPORT, RUNSTATS, and REORG INDEX, must know the object type that is to be processed before processing can begin. These utilities require that you specify an object type in addition to the LIST keyword (for example: REPORT

RECOVERY TABLESPACE LIST, RUNSTATS INDEX LIST, and REORG INDEX LIST). See the syntax diagrams for an individual utility for details.

Using the TEMPLATE utility with LISTDEF

Many utilities require output data sets. In those cases, you should use the TEMPLATE control statement to specify the naming convention and, optionally, the allocation parameters for each type of output data set. Templates, like lists, can be reused if the naming convention is robust enough to prevent duplicate data set names from being allocated. See Chapter 31, "TEMPLATE," on page 593 for more details about using templates.

In some cases you can use traditional JCL DD statements with LISTDEF lists, but this method is usually not practical unless you are processing small lists one object at a time.

Together, the LISTDEF and TEMPLATE utilities enable faster development of utility job streams, and require fewer modifications when the underlying list of database objects change.

Using the OPTIONS utility with LISTDEF

Use the following three functions of the OPTIONS utility in conjunction with the LISTDEF utility when needed:

OPTIONS PREVIEW
Enables you to preview the list contents before actual processing.

OPTIONS ITEMERROR
Enables you to alter the handling of errors that might occur during list processing.

OPTIONS LISTDEFDD
Enables you to identify a LISTDEF library. The default is LISTDEFDD. For more information about LISTDEF libraries, see "Creating LISTDEF libraries" on page 185.

Terminating or restarting LISTDEF

You can terminate a LISTDEF utility job by using the TERM UTILITY command if you submitted the job or have SYSOPR, SYSCTRL, or SYSADM authority.

You can restart a LISTDEF utility job, but it starts from the beginning again. Use caution when changing LISTDEF lists prior to a restart. When DB2 restarts list processing, it uses a saved copy of the list. Modifying the LISTDEF list that is referred to by the stopped utility has no effect. Only control statements that follow the stopped utility are affected. For guidance in restarting online utilities, see "Restarting an online utility" on page 43.

Concurrency and compatibility for LISTDEF

LISTDEF is a control statement that is used to set up an environment for another utility to follow. The LISTDEF list is stored until it is referenced by a specific utility. When referenced by an utility, the list expands. At that time, the concurrency and compatibility restrictions of that utility apply, with the additional restriction that the catalog tables that are necessary to expand the list must be available for read-only access.

List processing limitations: Although DB2 does not limit the number of objects that a list can contain, be aware that if your list is too large, the utility might fail with an error or abend in either DB2 or another program. These errors or abends can be caused by storage limitations, limitations of the operating system, or other restrictions imposed by either DB2 or non-DB2 programs. Whether such a failure occurs depends on many factors including, but not limited to the following items:

- The amount of available storage in both the utility batch and DBM1 address spaces
- The utility that is running.
- The type and number of other utilities that are running at the same time.
- The specific combination of keywords and operands of all the utilities that are running

Recommendation: If you receive a failure that you suspect is caused by running a utility on a list that is too large, divide your list into smaller lists and run the utility or utilities in separate job steps on the smaller lists until they run successfully.

Sample LISTDEF control statements

Example 1: Defining a list of objects. The following control statement defines a list that includes the following objects:

- Table space DBLT0301.TLLT031A
- Index space DBLT0301.IXlT031A
- Table space DBLT0301.IPLT031C
- Table space that contains ADMF001.TBLT032A_1

The name of the list is NAME1. This list can be referenced by any subsequent utility statements.

```
LISTDEF NAME1 INCLUDE TABLESPACE DBLT0301.TLLT031A
              INCLUDE INDEXSPACE DBLT0301.IXLT031A
              INCLUDE TABLESPACE DBLT0301.TPLT031C
              INCLUDE TABLE  ADMF001.TBLT032A_1
```

Example 2: Defining a list of all objects in a database. The following control statement defines a list (EXAMPLE2) that includes all table spaces and all index spaces in the PAYROLL database.

```
LISTDEF EXAMPLE2 INCLUDE TABLESPACES DATABASE PAYROLL
                 INCLUDE INDEXSPACES DATABASE PAYROLL
```

Example 3: Using pattern-matching characters. The following control statement defines a list (PAYROLL) that includes the following objects:

- All table spaces in the PAYROLL database, except for any table spaces whose names begin with TEMP.
- All index spaces in the PAYROLL database that end with IX, except for those index spaces that begin with TMPIX.

The subsequent COPY utility control statement processes this list.

```
LISTDEF PAYROLL INCLUDE TABLESPACE PAYROLL.*
                EXCLUDE TABLESPACE PAYROLL.TEMP*
                INCLUDE INDEXSPACE PAYROLL.*IX
                EXCLUDE INDEXSPACE PAYROLL.TMPIX*
COPY LIST PAYROLL ...
```

Example 4: Defining a list of partitions and nonpartitioned table spaces. The following control statement defines a list (EXAMPLE4) that includes one entry for

each partition of the qualifying partitioned table spaces and one entry for each qualifying nonpartitioned table space. The table spaces must satisfy the PAY*.* name pattern.

```
LISTDEF EXAMPLE4 INCLUDE TABLESPACE PAY*.* PARTLEVEL
```

Assume that three table spaces qualify. Of these table spaces, two are partitioned table spaces (PAY2.DEPTA and PAY2.DEPTF) that each have three partitions and one is a nonpartitioned table space (PAY1.COMP). In this case, the EXAMPLE4 list includes the following items:

- PAY2.DEPTA partition 1
- PAY2.DEPTA partition 2
- PAY2.DEPTA partition 3
- PAY2.DEPTF partition 1
- PAY2.DEPTF partition 2
- PAY2.DEPTF partition 3
- PAY1.COMP

If you specified PARTLEVEL(2) instead of PARTLEVEL, the EXAMPLE4 list includes the following items:

- PAY2.DEPTA partition 2
- PAY2.DEPTF partition 2
- PAY1.COMP

If you specified PARTLEVEL(0) instead of PARTLEVEL, the EXAMPLE4 list includes only PAY1.COMP.

Example 5: Defining a list of COPY YES indexes. The following control statement defines a list (EXAMPLE5) that includes related index spaces from the referenced list (EXAMPLE4) that have been defined or altered to COPY YES.

```
LISTDEF EXAMPLE5 INCLUDE LIST EXAMPLE4 INDEXSPACES COPY YES
```

Example 6: Defining a list that includes all table space partitions except for one. The following control statement defines a list (EXAMPLE6) that includes all partitions of table space X, except for partition 12. The INCLUDE clause adds an entry for each partition, and the EXCLUDE clause removes the entry for partition 12.

```
LISTDEF EXAMPLE6 INCLUDE TABLESPACE X PARTLEVEL
                 EXCLUDE TABLESPACE X PARTLEVEL(12)
```

Note that if the PARTLEVEL keyword is not specified in both clauses, as in the following two sample statements, the INCLUDE and EXCLUDE items do not intersect. For example, in the following statement, table space X is included is included in the list in its entirety, not at the partition level. Therefore, partition 12 cannot be excluded.

```
LISTDEF EXAMPLE6 INCLUDE TABLESPACE X
                 EXCLUDE TABLESPACE X PARTLEVEL(12)
```

In the following sample statement, the list includes only partition 12 of table space X, so table space X in its entirety can not be excluded.

```
LISTDEF EXAMPLE6 INCLUDE TABLESPACE X PARTLEVEL(12)
                 EXCLUDE TABLESPACE X
```

Example 7: Defining a LISTDEF library and using a list in a QUIESCE job. In Figure 32, the first two LISTDEF control statements define the NAME1 and NAME2 lists. The NAME1 list is stored in a sequential data set (JULTU103.TCASE.DATA2), and the NAME2 list is stored in a member of a partitioned data set (JULTU103.TCASE.DATA3(MEM1)). These output data sets are identified by the SYSUT2 DD statements (in the JCL for the CREATE1 and CREATE2 jobs).

The LISTLIB DD statement (in the JCL for the QUIESCE job) defines a LISTDEF library. When you define a LISTDEF library, you give a name to a group of data sets that contain LISTDEF statements. In this case, the library is to include the following data sets:

- The sequential data set JULTU103.TCASE.DATA2 (which includes the NAME1 list)
- The MEM1 member of the partitioned data set JULTU103.TCASE.DATA3 (which includes the NAME2 list).

Defining such a library enables you to subsequently refer to a group of LISTDEF statements with a single reference.

The OPTIONS utility control statement in this example specifies that the library that is identified by the LISTLIB DD statement is to be used as the default LISTDEF definition library. This declaration means that for any referenced lists, DB2 is to first search SYSIN for the list definition. If DB2 does not find the list definition in SYSIN, it is to search any data sets that are included in the LISTLIB LISTDEF library.

The last LISTDEF statement defines the NAME3 list. This list includes all objects in the NAME1 and NAME2 lists, except for three table spaces (TSLT032B, TSLT031B, TSLT032C). Because the NAME1 and NAME2 lists are not included in SYSIN, DB2 searches the default LISTDEF library (LISTLIB) to find them.

Finally, the QUIESCE utility control statement specifies this list of objects (NAME3) for which DB2 is to establish a quiesce point.

```
//CREATE1  JOB 'USER=NAME',CLASS=A,...
//*-------------------------------------------------------
//* Create an input data set.
//*-------------------------------------------------------
//LOAD1    EXEC PGM=IEBGENER
//SYSPRINT DD DUMMY
//SYSIN    DD DUMMY
//SYSUT2   DD DSN=JULTU103.TCASE.DATA2,
//         DISP=(NEW,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400)
//SYSUT1   DD *
  LISTDEF NAME1 INCLUDE TABLESPACE DBLT0301.TLLT031A
                INCLUDE TABLESPACE DBLT0301.TSLT031B
/*
//CREATE2  JOB 'USER=NAME',CLASS=A,...
//*-------------------------------------------------------
//* Create an input data set.
//*-------------------------------------------------------
//CRECNTL  EXEC PGM=IEFBR14
//CNTL     DD  DSN=JULTU103.TCASE.DATA3,UNIT=SYSDA,
//             VOL=SER=SCR03,
//             SPACE=(TRK,(2,2,2)),DCB=(DSORG=PO,
//             LRECL=80,RECFM=FB,BLKSIZE=4560),
//             DISP=(NEW,CATLG,CATLG)
/*
//*-------------------------------------------------------
//* Create member of input data set.
//*-------------------------------------------------------
//FILLCNTL EXEC PGM=IEBUPDTE
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DSN=JULTU103.TCASE.DATA3,DISP=OLD
//SYSUT2   DD  DSN=JULTU103.TCASE.DATA3,DISP=OLD
//SYSIN  DD  DATA
./ ADD NAME=MEM1
  LISTDEF NAME2 INCLUDE TABLESPACE DBLT0302.TLLT032A
                INCLUDE TABLESPACE DBLT0302.TSLT032B
                INCLUDE TABLESPACE DBLT0302.TPLT032C
./ ENDUP
/*
```

Figure 32. Example of building a LISTDEF library and then running the QUIESCE utility (Part 1 of 2)

```
//QUIESCE  JOB 'USER=NAME',CLASS=A,...
//*******************************************************
//* QUIESCE LISTDEF DD LILSTDEF data sets
//*******************************************************
//STEP1    EXEC DSNUPROC,UID='JULTU103.QUIESC2',
//            UTPROC='',SYSTEM='SSTR'
//LISTLIB  DD DSN=JULTU103.TCASE.DATA2,DISP=SHR
//         DD DSN=JULTU103.TCASE.DATA3(MEM1),DISP=SHR
//SYSIN    DD *
    OPTIONS LISTDEFDD LISTLIB
    LISTDEF NAME3 INCLUDE LIST NAME1
                  INCLUDE LIST NAME2
                  EXCLUDE TABLESPACE DBLT0302.TSLT032B
                  EXCLUDE TABLESPACE DBLT0301.TSLT031B
                  EXCLUDE TABLESPACE DBLT0302.TPLT032C
    QUIESCE LIST NAME3
/*
```

Figure 32. Example of building a LISTDEF library and then running the QUIESCE utility (Part 2 of 2)

Example 8: Defining a list that includes related objects. The following LISTDEF control statement defines a list (EXAMPLE8) that includes table space DBLT0101.TPLT011C and all objects that are referentially related to it. Only base table spaces are included in the list. The subsequent RECOVER utility control statement specifies that all objects in the EXAMPLE8 list are to be recovered.

```
//STEP2    EXEC DSNUPROC,UID='JULTU101.RECOVE5',
//            UTPROC='',SYSTEM='SSTR'
//SYSIN    DD *
    LISTDEF EXAMPLE8 INCLUDE TABLESPACE DBLT0101.TPLT011C RI BASE
    RECOVER LIST EXAMPLE8
/*
```

Chapter 16. LOAD

Use LOAD to load one or more tables of a table space. The LOAD utility loads records into the tables and builds or extends any indexes that are defined on them. If the table space already contains data, you can choose whether you want to add the new data to the existing data or replace the existing data. The loaded data is processed by any edit or validation routine that is associated with the table, and any field procedure that is associated with any column of the table. The LOAD utility ignores and does not enforce informational referential constraints.

For a diagram of LOAD syntax and a description of available options, see "Syntax and options of the LOAD control statement" on page 195. For detailed guidance on running this utility, see "Instructions for running LOAD" on page 234.

Output: LOAD DATA generates one or more of the following forms of output:

- A loaded table space or partition.
- A discard file of rejected records.
- A summary report of errors that were encountered during processing; this report is generated only if you specify ENFORCE CONSTRAINTS or if the LOAD involves unique indexes.

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorizations:

- Ownership of the table
- LOAD privilege for the database
- DBADM or DBCTRL authority for the database
- SYSCTRL or SYSADM authority

LOAD operates on a table space level, so you must have authority for all tables in the table space when you perform LOAD.

To run LOAD STATISTICS, the privilege set must include STATS authority on the database. To run LOAD STATISTICS TABLE ALL REPORT YES, the privilege set must also include the SELECT privilege on the tables required.

If you use RACF access control with multilevel security and LOAD is to process a table space that contains a table that has multilevel security with row-level granularity, you must be identified to RACF and have an accessible valid security label. You must also meet the following authorization requirements:

- To replace an entire table space with LOAD REPLACE, you must have the write-down privilege unless write-down rules are not in effect.
- You must have the write-down privilege to specify values for the security label columns, unless write-down rules are not in effect. If these rules are in effect and you do not have write-down privilege, DB2 assigns your security label as the value for the security label column for the rows that you are loading.

For more information about multilevel security and security labels, see Part 3 of *DB2 Administration Guide*.

Execution phases of LOAD: The LOAD utility operates in the phases that are listed in Table 27 on page 194.

Table 27. LOAD phases of execution

Phase	Description
UTILINIT	Performs initialization.
RELOAD	Loads record types and writes temporary file records for indexes and foreign keys. RELOAD makes one pass through the sequential input data set. Check constraints are checked for each row. Internal commits provide commit points at which to restart in case operation should halt in this phase. RELOAD creates inline copies if you specified the COPYDDN or RECOVERYDDN keywords. A subtask is started at the beginning of the RELOAD phase to sort the keys. The sort subtask initializes and waits for the main RELOAD phase to pass its keys to SORT. RELOAD loads the data, extracts the keys, and passes them in memory for sorting. At the end of the RELOAD phase, the last key is passed to SORT, and record sorting completes. Note that load partition parallelism starts subtasks. PREFORMAT for table spaces occurs at the end of the RELOAD phase.
SORT	Sorts temporary file records before creating indexes or validating referential constraints, if indexes or foreign keys exist. The SORT phase is skipped if all the following conditions apply for the data that is processed during the RELOAD phase: • Each table has no more than one key. • All keys are the same type (index key only, indexed foreign key, or foreign key only). • The data that is being loaded or reloaded is in key order (if a key exists). If the key is an index key only and the index is a data-partitioned secondary index, the data is considered to be in order if the data is grouped by partition and ordered within partition by key value. If the key in question is an indexed foreign key and the index is a data-partitioned secondary index, the data is never considered to be in order. • The data that is being loaded or reloaded is grouped by table, and each input record is loaded into one table only. SORT passes the sorted keys in memory to the BUILD phase, which builds the indexes.
BUILD	Creates indexes from temporary file records for all indexes that are defined on the loaded tables. Build also detects duplicate keys. PREFORMAT for indexes occurs at the end of the BUILD phase.
SORTBLD	Performs all activities that normally occur in both the SORT and BUILD phases, if you specify a parallel index build.
INDEXVAL	Corrects unique index violations from the information in SYSERR, if any exist.
ENFORCE	Checks referential constraints, except informational referential constraints, and corrects violations. Information about violations of referential constraints is stored in SYSERR.
DISCARD	Copies records that cause errors from the input data set to the discard data set.
REPORT	Generates a summary report, if you specified ENFORCE CONSTRAINT or if load index validation is performed. The report is sent to SYSPRINT.
UTILTERM	Performs cleanup.

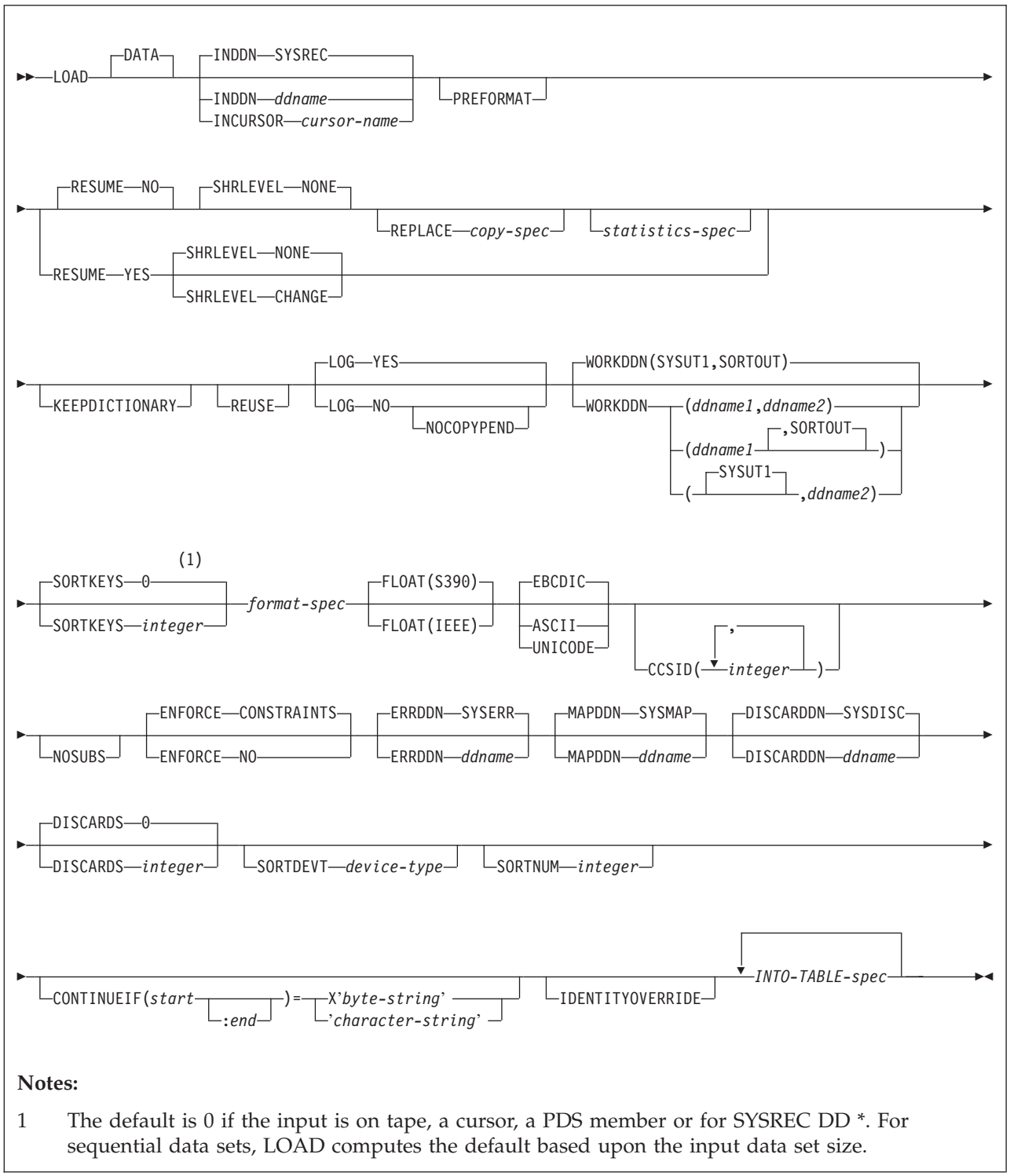
The following topics provide additional information:

- "Syntax and options of the LOAD control statement" on page 195
- "Instructions for running LOAD" on page 234

Syntax and options of the LOAD control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

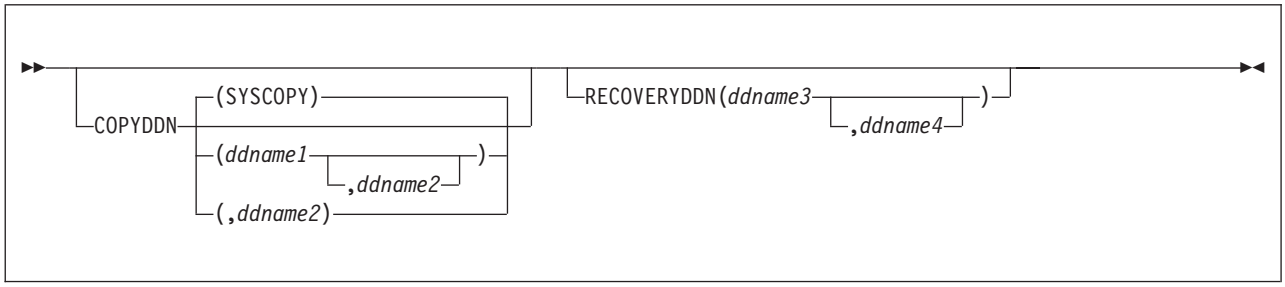
Syntax diagram



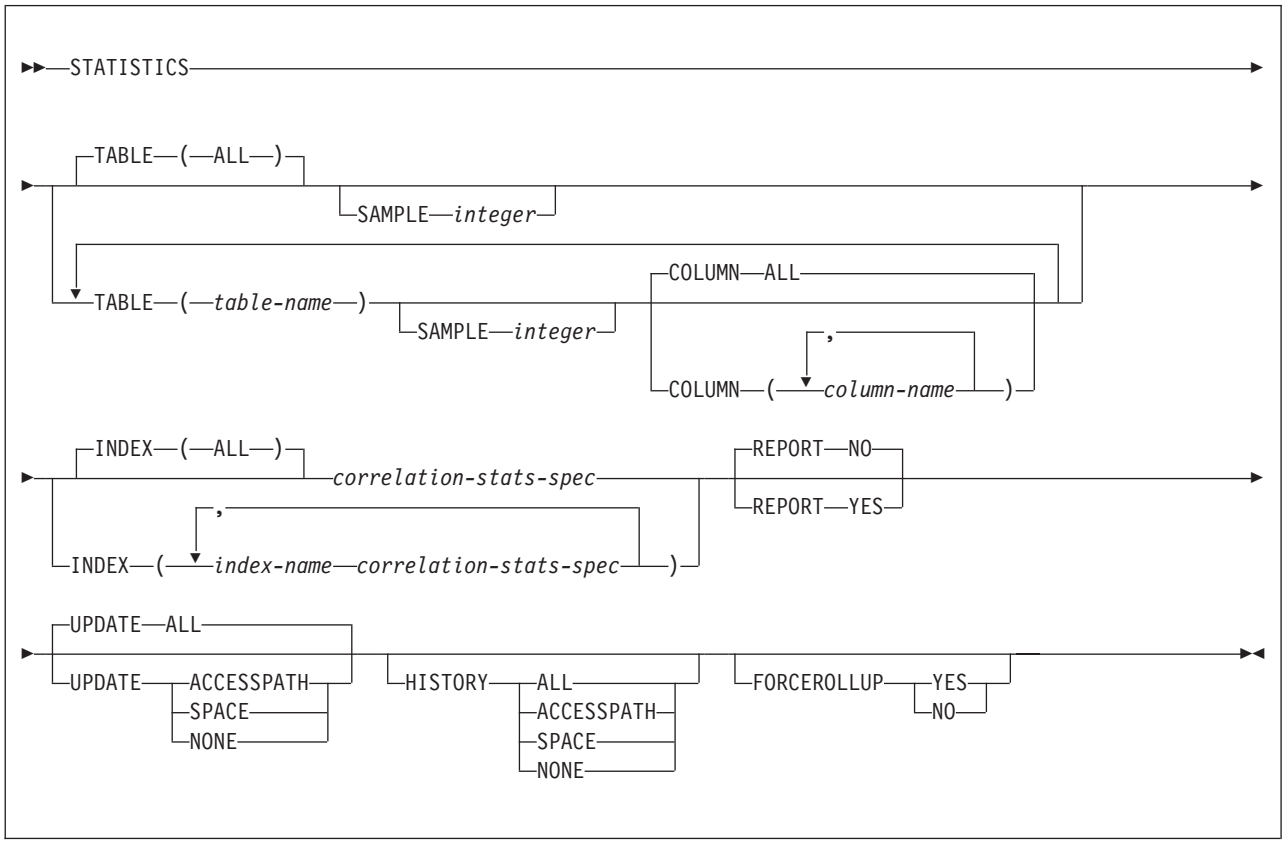
Notes:

1 The default is 0 if the input is on tape, a cursor, a PDS member or for SYSREC DD *. For sequential data sets, LOAD computes the default based upon the input data set size.

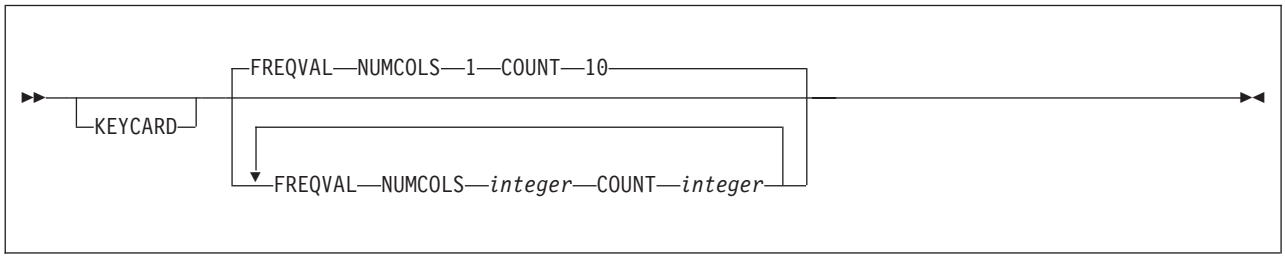
copy-spec:



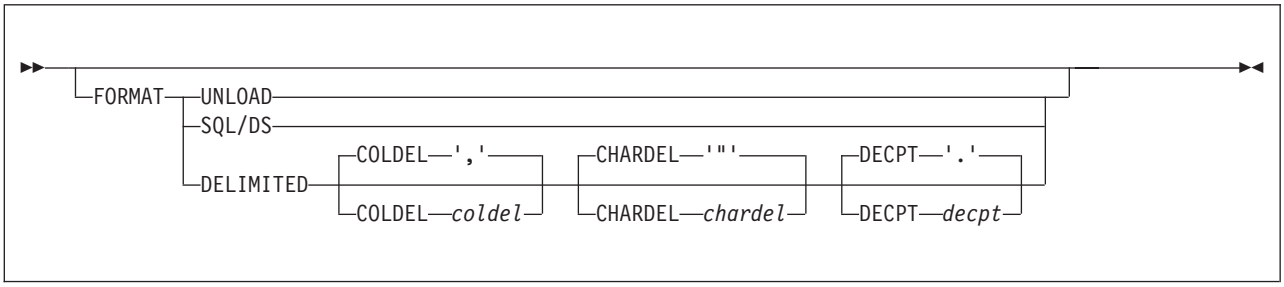
statistics-spec:



correlation-stats-spec:



format-spec:



INTO-TABLE-spec:

For the syntax diagram and the option descriptions of the into-table specification, see "INTO-TABLE-spec" on page 213.

Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

DATA Specifies that data is to be loaded. This keyword is optional and is used for clarity only.

INDDN *ddname*
Specifies the data definition (DD) statement or template that identifies the input data set for the partition. The record format for the input data set must be fixed-length or variable-length. The data set must be readable by the basic sequential access method (BSAM).

The *ddname* is the name of the input data set. The **default** is **SYSREC**.

INCURSOR *cursor-name*
Specifies the cursor for the input data set. You must declare the cursor before it is used by the LOAD utility. Use the EXEC SQL utility control statement to define the cursor. You cannot load data into the same table on which you defined the cursor.

The specified cursor can be used with the DB2 UDB family cross-loader function, which enables you to load data from any DRDA-compliant remote server. For more information about using the cross-loader function, see "Loading data by using the cross-loader function" on page 251.

cursor-name is the cursor name. Cursor names that are specified with the LOAD utility cannot be longer than eight characters.

You cannot use the INCURSOR option with the following options:
- SHRLEVEL CHANGE
- NOSUBS
- FORMAT UNLOAD
- FORMAT SQL/DS
- CONTINUEIF
- WHEN

In addition, you cannot specify field specifications or use discard processing with the INCURSOR option.

PREFORMAT
Specifies that the remaining pages are preformatted up to the

high-allocated RBA in the table space and index spaces that are associated with the table that is specified in *table-name*. The preformatting occurs after the data has been loaded and the indexes are built.

PREFORMAT can operate on an entire table space and its index spaces, or on a partition of a partitioned table space and on the corresponding partitions of partitioned indexes, if any exist. Specifying LOAD PREFORMAT (rather than PART *integer* PREFORMAT) tells LOAD to serialize at the table space level, which can inhibit concurrent processing of separate partitions. If you want to serialize at the partition level, specify PART *integer* PREFORMAT. See "Option descriptions for INTO TABLE" on page 216 for information about specifying PREFORMAT at the partition level.

RESUME

Indicates whether records are to be loaded into an empty or non-empty table space. For nonsegmented table spaces, space is not reused for rows that have been marked as deleted or for rows of dropped tables.

Important: Specifying LOAD RESUME (rather than PART *integer* RESUME) tells LOAD to serialize on the entire table space, which can inhibit concurrent processing of separate partitions. If you want to process other partitions concurrently, use "INTO-TABLE-spec" on page 213 to specify PART *integer* RESUME.

NO

Loads records into an empty table space. If the table space is not empty, and you have not used REPLACE, a message is issued and the utility job step terminates with a job step condition code of 8.

For nonsegmented table spaces that contain deleted rows or rows of dropped tables, using the REPLACE keyword provides increased efficiency.

The **default** is **NO**, unless you override it with PART *integer* RESUME YES.

YES

Loads records into a non-empty table space. If the table space is empty, a warning message is issued, but the table space is loaded. Loading begins at the current end of data in the table space. Space is not reused for rows that are marked as deleted or for rows of dropped tables.

LOAD RESUME SHRLEVEL CHANGE activates the before triggers and after triggers for each row that is loaded.

SHRLEVEL

Specifies the extent to which applications can concurrently access the table space or partition during the LOAD utility job. The following parameter values are listed in order of increasing extent of allowed concurrent access.

NONE

Specifies that applications have no concurrent access to the table space or partition.

The **default** is **NONE**.

CHANGE

Specifies that applications can concurrently read from and write to the table space or partition into which LOAD is loading data. If you specify SHRLEVEL CHANGE, you cannot specify the following

parameters: INCURSOR, RESUME NO, REPLACE, KEEPDICTIONARY, LOG NO, ENFORCE NO, STATISTICS, COPYDDN, RECOVERYDDN, MAPDDN, PREFORMAT, REUSE, or PART *integer* REPLACE.

For a partition-directed LOAD, if you specify SHRLEVEL CHANGE, only RESUME YES can be specified or inherited from the LOAD statement.

LOAD SHRLEVEL CHANGE does not perform the SORT, BUILD, SORTBLD, INDEXVAL, ENFORCE, or REPORT phases, and the compatibility and concurrency considerations differ.

A LOAD SHRLEVEL CHANGE job functions like a mass INSERT. Whereas a regular LOAD job drains the entire table space, LOAD SHRLEVEL CHANGE functions like an INSERT statement and uses claims when accessing an object.

Normally, a LOAD RESUME YES job loads the records at the end of the already existing records. However, for a LOAD RESUME YES job with the SHRLEVEL CHANGE option, the utility tries to insert the new records in available free space as close to the clustering order as possible. This LOAD job does not create any additional free pages. If you insert a lot of records, these records are likely to be stored out of clustering order. In this case, you should run the REORG TABLESPACE utility after loading the records.

When an identity column exists in the table being loaded, performance can be improved by specifying the CACHE attribute for the identity column.

Recommendation: If you have loaded a lot of records, run RUNSTATS SHRLEVEL CHANGE UPDATE SPACE and then a conditional REORG.

Log records that DB2 creates during LOAD SHRLEVEL CHANGE can be used by DB2 DataPropagator, if the tables that are being loaded are defined with DATA CAPTURE CHANGES.

Note that before and after row triggers are activated for SHRLEVEL CHANGE but not for SHRLEVEL NONE. Statement triggers for each row are also activated for SHRLEVEL CHANGE but not for SHRLEVEL NONE.

REPLACE

Indicates whether the table space and all its indexes need to be reset to empty before records are loaded. With this option, the newly loaded rows replace **all** existing rows of all tables in the table space, not just those of the table that you are loading. For DB2 STOGROUP-defined data sets, the data set is deleted and redefined with this option, unless you also specified the REUSE option. You must have LOAD authority for all tables in the table space where you perform LOAD REPLACE. If you attempt a LOAD REPLACE without this authority, you get an error message.

You cannot use REPLACE with the PART *integer* REPLACE option of INTO TABLE; you must either replace an entire table space by using the REPLACE option or replace a single partition by using the PART *integer* REPLACE option of INTO TABLE.

Specifying LOAD REPLACE (rather than PART *integer* REPLACE) tells LOAD to serialize at the table space level. If you want to serialize at the

partition level, specify PART *integer* REPLACE. See the information about specifying REPLACE at the partition level under the keyword descriptions for INTO TABLE.

COPYDDN (*ddname1,ddname2***)**
Specifies the DD statements for the primary (*ddname1*) and backup (*ddname2*) copy data sets for the image copy.

ddname is the DD name.

The **default** is **SYSCOPY** for the primary copy. No default exists for the backup copy.

The COPYDDN keyword can be specified only with REPLACE. A full image copy data set (SHRLEVEL REFERENCE) is created for the table or partitions that are specified when LOAD executes. The table space or partition for which an image copy is produced is not placed in COPY-pending status.

Image copies that are taken during LOAD REPLACE are not recommended for use with RECOVER TOCOPY because these image copies might contain unique index violations or referential constraint violations.

Using COPYDDN when loading a table with LOB columns does not create a copy of any index or LOB table space. You must perform these tasks separately.

The COPYDDN keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, “TEMPLATE,” on page 593.

RECOVERYDDN *ddname3,ddname4*
Specifies the DD statements for the primary (*ddname3*) and backup (*ddname4*) copy data sets for the image copy at the recovery site.

ddname is the DD name.

You cannot have duplicate image copy data sets. The same rules apply for RECOVERYDDN and COPYDDN.

The RECOVERYDDN keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, “TEMPLATE,” on page 593.

STATISTICS
Specifies the gathering of statistics for a table space, index, or both; the statistics are stored in the DB2 catalog.

If you specify the STATISTICS keyword with no other *statistics-spec* or *correlation-stats-spec* options, DB2 gathers only table space statistics. Statistics are collected on a base table space, but not on a LOB table space.

Restriction: If you specify STATISTICS for encrypted data, DB2 might not provide useful statistics on this data.

TABLE
Specifies the table for which column information is to be gathered. All tables must belong to the table space that is specified in the TABLESPACE option.

(ALL)
Specifies that information is to be gathered for all columns of all tables in the table space. The **default** is **ALL**.

(*table-name***)**
Specifies the tables for which column information is to be gathered. If you omit the qualifier, the user identifier for the utility job is used. Enclose the table name in quotation marks if the name contains a blank.

If you specify more than one table, you must repeat the TABLE option. Multiple TABLE options must be specified entirely before or after any INDEX keyword that may also be specified. For example, the INDEX keyword may not be specified between any two TABLE keywords.

SAMPLE *integer*
Indicates the percentage of rows that LOAD is to sample when collecting non-indexed column statistics. You can specify any value from 1 through 100. The **default** is **25**.

COLUMN
Specifies the columns for which column information is to be gathered.

You can specify this option only if you specify the particular tables for which statistics are to be gathered (TABLE (table-name)). If you specify particular tables and do not specify the COLUMN option, the default, COLUMN(ALL), is used. If you do not specify a particular table when using the TABLE option, you cannot specify the COLUMN option; however, COLUMN(ALL) is assumed.

(ALL)
Specifies that statistics are to be gathered for all columns in the table. The **default** is **ALL**.

(*column-name***, ...)**
Specifies the columns for which statistics are to be gathered.

You can specify a list of column names; the maximum is 10. If you specify more than one column, separate each name with a comma.

INDEX
Specifies indexes for which information is to be gathered. Column information is gathered for the first column of the index. All the indexes must be associated with the same table space, which must be the table space that is specified in the TABLESPACE option.

(ALL)
Specifies that the column information is to be gathered for all indexes that are defined on tables in the table space.

(*index-name***)**
Specifies the indexes for which information is to be gathered. Enclose the index name in quotation marks if the name contains a blank.

KEYCARD
Requests the collection of all distinct values in all of the 1 to *n* key column combinations for the specified indexes. *n* is the number of columns in the index.

FREQVAL
Controls the collection of frequent-value statistics. If you specify FREQVAL, it must be followed by two additional keywords:

NUMCOLS
Indicates the number of key columns that are to be concatenated together when collecting frequent values from the specified index. Specifying '3' means that frequent values are to be collected on the concatenation of the first three key columns. The **default** is **1**, which means that DB2 collects frequent values on the first key column of the index.

COUNT
Indicates the number of frequent values that are to be collected. Specifying '15' means that DB2 collects 15 frequent values from the specified key columns. The **default** is **10**.

REPORT
Indicates whether a set of messages is to be generated to report the collected statistics.

NO
Indicates that the set of messages is not sent to SYSPRINT as output.

The **default** is **NO**.

YES
Indicates that the set of messages is sent to SYSPRINT as output. The generated messages are dependent on the combination of keywords (such as TABLESPACE, INDEX, TABLE, and COLUMN) that are specified with the RUNSTATS utility. However, these messages are not dependent on the specification of the UPDATE option. REPORT YES always generates a report of SPACE and ACCESSPATH statistics.

UPDATE
Indicates whether the collected statistics are to be inserted into the catalog tables. UPDATE also allows you to select statistics that are used for access path selection or statistics that are used by database administrators.

ALL Indicates that all collected statistics are to be updated in the catalog.

The **default** is **ALL**.

ACCESSPATH
Indicates that updates are to be made only to the catalog table columns that provide statistics that are used for access path selection.

SPACE
Indicates that updates are to be made only to the catalog table columns that provide statistics to help database administrators assess the status of a particular table space or index.

NONE
Indicates that no catalog tables are to be updated with the collected statistics. This option is valid only when REPORT YES is specified.

HISTORY

Records all catalog table inserts or updates to the catalog history tables.

The default is supplied by the value that is specified in STATISTICS HISTORY on panel DSNTIPO.

ALL Indicates that all collected statistics are to be updated in the catalog history tables.

ACCESSPATH

Indicates that updates are to be made only to the catalog history table columns that provide statistics that are used for access path selection.

SPACE

Indicates that only space-related catalog statistics are to be updated in catalog history tables.

NONE

Indicates that no catalog history tables are to be updated with the collected statistics.

FORCEROLLUP

Specifies whether aggregation or rollup of statistics is to take place when RUNSTATS is executed even if some parts are empty. This keyword enables the optimizer to select the best access path.

YES Indicates that forced aggregation or rollup processing is to be done, even though some parts might not contain data.

NO Indicates that aggregation or rollup is to be done only if data is available for all parts.

If data is not available for all parts, DSNU623I message is issued if the installation value for STATISTICS ROLLUP on panel DSNTIPO is set to NO.

KEEPDICTIONARY

Prevents the LOAD utility from building a new compression dictionary. LOAD retains the current compression dictionary and uses it for compressing the input data. This option eliminates the cost that is associated with building a new dictionary.

This keyword is valid only if the table space that is being loaded has the COMPRESS YES attribute.

If the table space or partition is empty, DB2 performs one of these actions:
- DB2 builds a dictionary if a compression dictionary does not exist.
- DB2 keeps the dictionary if a compression dictionary exists.

If RESUME NO and REPLACE are specified when the table space or partition is not empty, DB2 performs the same actions as it does when the table space or partition is empty.

If the table space or partition is not empty and RESUME YES is specified, DB2 performs one of these actions:
- DB2 does not build a dictionary if a compression dictionary does not exist.
- DB2 keeps the dictionary if a compression dictionary exists.

Note: You must use KEEPDICTIONARY to ensure that the compression dictionary is maintained.

For information regarding data compression, see Part 5 (Volume 2) of *DB2 Administration Guide*.

REUSE

Specifies (when used with REPLACE) that LOAD is to logically reset and reuse DB2-managed data sets without deleting and redefining them. If you do not specify REUSE, DB2 deletes and redefines DB2-managed data sets to reset them.

REUSE must be accompanied by REPLACE to do the logical reset for all data sets. However, if you specify REUSE for the table space and REPLACE only at the partition level, only the replaced partitions are logically reset. See the description of REUSE in "INTO-TABLE-spec" on page 213 for information about specifying REUSE for individual partitions.

If a data set has multiple extents, the extents are not released if you specify the REUSE parameter.

LOG Indicates whether logging is to occur during the RELOAD phase of the load process.

YES

Specifies normal logging during the load process. All records that are loaded are logged. The **default** is **YES**.

NO

Specifies no logging of data during the load process. The NO option sets the COPY-pending restriction against the table space or partition that the loaded table resides in. No table or partition in the table space can be updated by SQL until the restriction is removed. For ways to remove the restriction, see "Resetting COPY-pending status" on page 269.

If you load a single partition of a partitioned table space and the table space has a secondary index, some logging might occur during the build phase as DB2 logs any changes to the index structure. This logging allows recoverability of the secondary index in case an abend occurs, and it also allows concurrency.

A LOB table space affects logging while DB2 loads a LOB column regardless of whether the LOB table space was defined with LOG YES or LOG NO. See Table 40 on page 263 for more information.

NOCOPYPEND

Specifies that LOAD is not to set the table space in the COPY-pending status, even though LOG NO was specified. A NOCOPYPEND specification does not turn on or change any informational COPY-pending (ICOPY) status for indexes. A NOCOPYPEND specification will not turn off any COPY-pending status that was set prior to the LOAD. Normal completion of a LOAD LOG NO NOCOPYPEND job returns a 0 code if no other errors or warnings exist.

DB2 ignores a NOCOPYPEND specification if you also specified COPYDDN to make a local-site inline image copy during the LOAD.

Attention: Specify the NOCOPYPEND option only if the data in the table space can be easily recreated by another LOAD job if the data is lost. If you do not take an image copy following the LOAD, you cannot recover the table space by using the RECOVER utility and you might lose data.

WORKDDN (*ddname1***,***ddname2***)**

Specifies the DD statements for the temporary work file for sort input and sort output. Temporary work files for sort input and output are required if the LOAD involves tables with indexes.

ddname1 is the DD name for the temporary work file for sort input. The **default** is **SYSUT1**.

ddname2 is the DD name for the temporary work file for sort output. The **default** is **SORTOUT**.

The WORKDDN keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

SORTKEYS *integer*

Specifies that index keys are to be sorted in parallel during the SORTBLD phase to improve performance. Optionally, you can specify a value for *integer* to provide an estimate of the number of index keys that are to be sorted. Integer must be a positive integer between 0 and 562 949 953 421 311. The **default** is **0** if any of the following conditions are true:

- The target table has no index and SHRLEVEL is NONE.
- The target table has one index.
- The input is on tape, a cursor, a PDS member, or for SYSREC DD *.

For sequential data sets, LOAD computes an estimate based upon the input data set size.

For more information about sorting keys, see "Improved performance with SORTKEYS" on page 253 and "Building indexes in parallel for LOAD" on page 258.

FORMAT

Identifies the format of the input record. If you use FORMAT UNLOAD or FORMAT SQL/DS, it uniquely determines the format of the input, and no field specifications are allowed in an INTO TABLE option.

If you omit FORMAT, the format of the input data is determined by the rules for field specifications that are described in"Option descriptions for INTO TABLE" on page 216. If you specify FORMAT DELIMITED, the format of the input data is determined by the rules that are described in Appendix F, "Delimited file format," on page 899.

UNLOAD

Specifies that the input record format is compatible with the DB2 unload format. (The DB2 unload format is the result of REORG with the UNLOAD ONLY option.)

Input records that were unloaded by the REORG utility are loaded into the tables from which they were unloaded, if an INTO TABLE

option specifies each table. Do not add columns or change column definitions of tables between the time you run REORG UNLOAD ONLY and LOAD FORMAT UNLOAD.

Any WHEN clause on the LOAD FORMAT UNLOAD statement is ignored; DB2 reloads the records into the same tables from which they were unloaded. Not allowing a WHEN clause with the FORMAT UNLOAD clause ensures that the input records are loaded into the proper tables. Input records that cannot be loaded are discarded.

If the DCB RECFM parameter is specified on the DD statement for the input data set, and the data set format has not been modified since the REORG UNLOAD (ONLY) operation, the record format must be variable (RECFM=V).

SQL/DS

Specifies that the input record format is compatible with the SQL/DS unload format. The data type of a column in the table that is to be loaded must be the same as the data type of the corresponding column in the SQL/DS table.

If the SQL/DS input contains rows for more than one table, the WHEN clause of the INTO TABLE option indicates which input records are to be loaded into which DB2 table.

For information about the correct DCB parameters to specify on the DD statement for the input data set, refer to *DB2 Server for VM: DBS Utility*.

LOAD cannot load SQL/DS strings that are longer than the DB2 limit. For information about DB2 limits, see Appendix A, "Limits in DB2 UDB for z/OS," on page 791.

SQL/DS data that has been unloaded to disk under DB2 Server for VSE & VM resides in a simulated z/OS-type data set with a record format of VBS. Consider this format when transferring the data to another system that is to be loaded into a DB2 table (for example, the DB2 Server for VSE & VM. FILEDEF must define it as a z/OS-type data set). Processing the data set as a standard CMS file puts the SQL/DS record type field at the wrong offset within the records; LOAD is unable to recognize them as valid SQL/DS input.

DELIMITED

Specifies that the input data file is in a delimited format. When data is in a delimited format, all fields in the input data set are character strings or external numeric values. In addition, each column in a delimited file is separated from the next column by a column delimiter character.

For each of the delimiter types that you can specify, you must ensure that the delimiter character is specified in the code page of the source data. The delimiter character can be specified as either a character or hex constant. For example, to specify '#' as the delimiter, you can specify either COLDEL '#' or COLDEL X'23'. If the utility statement is coded in a character type that is different from the input file, such as a utility statement that is coded in EBCDIC and input data that is in Unicode, you should specify the delimiter character in the utility statement as a hex constant, or the result can be unpredictable.

You cannot specify the same character for more than one type of delimiter (COLDEL, CHARDEL, and DECPT). For more information about delimiter restrictions, see "Loading delimited files" on page 244.

Unicode input data for FORMAT DELIMITED must be UTF-8, CCSID 1208.

If you specify the FORMAT DELIMITED option, you cannot use any of the following options:
- CONTINUEIF
- INCURSOR
- Multiple INTO TABLE statements
- WHEN

Also, LOAD ignores any specified POSITION statements within the LOAD utility control statement.

For more information about using delimited output and delimiter restrictions, see "Loading delimited files" on page 244. For more information about delimited files see Appendix F, "Delimited file format," on page 899.

COLDEL *coldel*
Specifies the column delimiter that is used in the input file. The **default** is a comma (,). For ASCII and UTF-8 data this is X'2C', and for EBCDIC data it is a X'6B'.

CHARDEL *chardel*
Specifies the character string delimiter that is used in the input file. The **default** is a double quotation mark ("). For ASCII and UTF-8 data this is X'22', and for EBCDIC data it is X'3F'.

To delimit character strings that contain the character string delimiter, repeat the character string delimiter where it is used in the character string. LOAD interprets any pair of character delimiters that are found between the enclosing character delimiters as a single character. For example, the phrase `"what a ""nice warm"" day"` is interpreted as `what a "nice warm" day`. The LOAD utility recognizes these character delimiter pairs for only CHAR, VARCHAR, and CLOB fields.

Character string delimiters are required only when the string contains the CHARDEL character. However, you can put the character string delimiters around other character strings. Data that has been unloaded in delimited format by the UNLOAD utility includes character string delimiters around all character strings.

DECPT*decpt*
Specifies the decimal point character that is used in the input file. The **default** is a period (.).

The default decimal point character is a period in a delimited file, X'2E' in an ASCII or Unicode UTF-8 file.

Note: When the DECPT value is used in DSNHDECP, that value replaces the default DECPT value.

FLOAT

Specifies that LOAD is to expect the designated format for floating point numbers.

(S390)

Specifies that LOAD is to expect that floating point numbers are provided in System/390 hexadecimal floating point (HFP) format. **(S390)** is the format that DB2 stores floating point numbers in. It is also the **default** if you do not explicitly specify the FLOAT keyword.

(IEEE)

Specifies that LOAD is to expect that floating point numbers are provided in IEEE binary floating point (BFP) format.

When you specify FLOAT(IEEE), DB2 converts the BFP data to HFP format as the data is being loaded into the DB2 table. If a conversion error occurs while DB2 is converting from BFP to HFP, DB2 places the record in the discard file.

FLOAT(IEEE) is mutually exclusive with any specification of the FORMAT keyword. If you specify both FLOAT(IEEE) and FORMAT, DB2 issues message DSNU070I.

BFP format is sometimes called IEEE floating point.

EBCDIC

Specifies that the input data file is EBCDIC. The **default** is **EBCDIC**.

ASCII Specifies that the input data file is ASCII. Numeric, date, time, and timestamp internal formats are not affected by the ASCII option.

UNICODE

Specifies that the input data file is Unicode. The UNICODE option does not affect the numeric internal formats.

CCSID

Specifies up to three coded character set identifiers (CCSIDs) for the input file. The first value specifies the CCSID for SBCS data that is found in the input file, the second value specifies the CCSID for mixed DBCS data, and the third value specifies the CCSID for DBCS data. If any of these values is specified as 0 or omitted, the CCSID of the corresponding data type in the input file is assumed to be the same as the installation default CCSID. If the input data is EBCDIC, the omitted CCSIDs are assumed to be the EBCDIC CCSIDs that are specified at installation, and if the input data is ASCII, the omitted CCSIDs are assumed to be the ASCII CCSIDs that are specified at installation. If the CCSIDs of the input data file do not match the CCSIDs of the table that is being loaded, the input data is converted to the table CCSIDs before being loaded.

integer is any valid CCSID specification.

If the input data is Unicode, the default CCSID values are the Unicode CCSIDs that are specified at system installation.

NOSUBS

Specifies that LOAD is not to accept substitution characters in a string.

Place a substitution character in a string when that string is being converted from ASCII to EBCDIC, or when the string is being converted from one CCSID to another. For example, this substitution occurs when a character (sometimes referred to as a code point) that exists in the source CCSID (code page) does not exist in the target CCSID (code page).

When you specify the NOSUBS option and the LOAD utility determines that a substitution character has been placed in a string as a result of a conversion, it performs one of the following actions:

- **If discard processing is active:** DB2 issues message DSNU310I and places the record in the discard file.
- **If discard processing is not active:** DB2 issues message DSNU334I, and the utility abnormally terminates.

ENFORCE

Specifies whether LOAD is to enforce check constraints and referential constraints, except informational referential constraints, which are not enforced.

CONSTRAINTS

Indicates that constraints are to be enforced. If LOAD detects a violation, it deletes the errant row and issues a message to identify it. If you specify this option and referential constraints exist, sort input and sort output data sets must be defined.

The **default** is **CONSTRAINTS**.

NO Indicates that constraints are not to be enforced. This option places the target table space in the CHECK-pending status if at least one referential constraint or check constraint is defined for the table.

ERRDDN *ddname*

Specifies the DD statement for a work data set that is being used during error processing. Information about errors that are encountered during processing is stored in this data set. A SYSERR data set is required if you request discard processing.

ddname is the DD name. The **default** is **SYSERR**.

The ERRDDN keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

MAPDDN *ddname*

Specifies the DD statement for a work data set that is to be used during error processing. The work data set is used to correlate the identifier of a table row with the input record that caused an error. A SYSMAP data set is required if you specify ENFORCE CONSTRAINTS and the tables have a referential relationship, or if you request discard processing when loading one or more tables that contain unique indexes.

ddname is the DD name. The **default** is **SYSMAP**.

The MAPDDN keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

DISCARDDN *ddname*

Specifies the DD statement for a discard data set that is to hold copies of records that are not loaded (for example, if they contain conversion errors). The discard data set also holds copies of records that are loaded and then

removed (due to unique index errors, or referential or check constraint violations). Flag input records for discarding during RELOAD, INDEXVAL, and ENFORCE phases. However, the discard data set is not written until the DISCARD phase when the flagged records are copied from the input data set to the discard data set. The discard data set must be a sequential data set that can be written to by BSAM, with the same record format, record length, and block size as the input data set.

ddname is the DD name. The **default** is **SYSDISC**.

If you omit the DISCARDDN option, the utility application program saves discarded records only if a SYSDISC DD statement is in the JCL input.

The DISCARDDN keyword is not supported if you use a BatchPipes file as
an input to LOAD, using INDDN name for TEMPLATE SUBSYS.

The DISCARDDN keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

DISCARDS *integer*

Specifies the maximum number of source records that are to be written on the discard data set. *integer* can range from 0 to 2147483647. If the discard maximum is reached, LOAD abnormally terminates, the discard data set is empty, and you cannot see which records were discarded. You can either restart the job with a larger limit, or terminate the utility.

DISCARDS 0 specifies that you do not want to set a maximum value. The entire input data set can be discarded.

The **default** is **DISCARDS 0**.

SORTDEVT *device-type*

Specifies the device type for temporary data sets that are to be dynamically allocated by DFSORT. You can specify any device type that is acceptable to the DYNALLOC parameter of the SORT or OPTION options for DFSORT.

A TEMPLATE specification does not dynamically allocate sort work data sets. The SORTDEVT keyword controls dynamic allocation of these data sets.

SORTNUM *integer*

Indicates the number of temporary data sets that are to be dynamically allocated by the sort application program.

integer is the number of temporary data sets that can range from 2 to 255.

If you omit SORTDEVT, SORTNUM is ignored. If you use SORTDEVT and omit SORTNUM, no value is passed to DFSORT. In this case, DFSORT uses its own default.

You need at least two sort work data sets for each sort. The SORTNUM
value applies to each sort invocation in the utility. For example, if there are
three indexes, SORTKEYS is specified, there are no constraints limiting
parallelism, and SORTNUM is specified as 8, then a total of 24 sort work
data sets will be allocated for a job.

Each sort work data set consumes both above the line and below the link
virtual storage, so if you specify too high a value for SORTNUM, the

utility may decrease the degree of parallelism due to virtual storage constraints, and possibly decreasing the degree down to one, meaning no parallelism.

Important: The SORTNUM keyword will not be considered if ZPARM UTSORTAL is set to YES and IGNSORTN is set to YES.

CONTINUEIF

Indicates that you want to be able to treat each input record as a portion of a larger record. After CONTINUEIF, write a condition in one of the following forms:

```
(start:end) = X'byte-string'
(start:end) = 'character-string'
```

If the condition is true in any record, the next record is concatenated with it before loading takes place. You can concatenate any number of records into a larger record, up to a maximum size of 32767 bytes.

Data in the input record can be in ASCII or Unicode format, but the utility control statement always interprets character constants as EBCDIC. To use CONTINUEIF with the ASCII or UNICODE option, you must code the condition by using the hexadecimal form, not the character-string form. For example, use `(1:1)=X'31'` rather than `(1:1)='1'`. As an alternative, you can code the control statements in UTF-8. See "Unicode character strings" on page 17for more information about hex notation and UTF-8.

(*start:end***)**

Specifies column numbers in the input record; the first column of the record is column 1. The two numbers tell the starting and ending columns of a continuation field in the input record.

Other field position specifications (such as those for WHEN, POSITION, or NULLIF) refer to the field position within the final assembled load record, not within the input record.

The continuation field is removed from the input record and is not part of the final load record.

If you omit *:end*, DB2 assumes that the length of the continuation field is the length of the byte string or character string. If you use *:end*, and the length of the resulting continuation field is not the same as the length of the byte string or character string, the shorter string is padded. Character strings are padded with blanks. Hexadecimal strings are padded with zeros.

X'*byte-string***'**

Specifies a string of hexadecimal characters. This byte-string value in the continuation field indicates that the next input record is a continuation of the current load record. Records with this byte-string value are concatenated until the value in the continuation field changes. For example, the following CONTINUEIF specification indicates that for any input records that have a value of X'FF'in column 72, LOAD is to concatenate that record with the next input record.

```
CONTINUEIF (72) = X'FF'
```

'*character-string***'**

Specifies a string of characters that has the same effect as X'*byte-string*'. For example, the following CONTINUEIF specification indicates that for any input records that have the string CC in columns 99 and 100, LOAD is to concatenate that record with the next input record.

```
CONTINUEIF (99:100) = 'CC'
```

IDENTITYOVERRIDE

Allows unloaded data to be reloaded into a GENERATED ALWAYS identity column of the same table using LOAD REPLACE or LOAD RESUME. When this option is used and input field specifications are coded, the identity column must be specified and NULLIF or DEFAULTIF is not allowed.

Specifying this option will allow LOAD INTO TABLE PART when the identity column is part of the partitioning index.

INTO-TABLE-spec

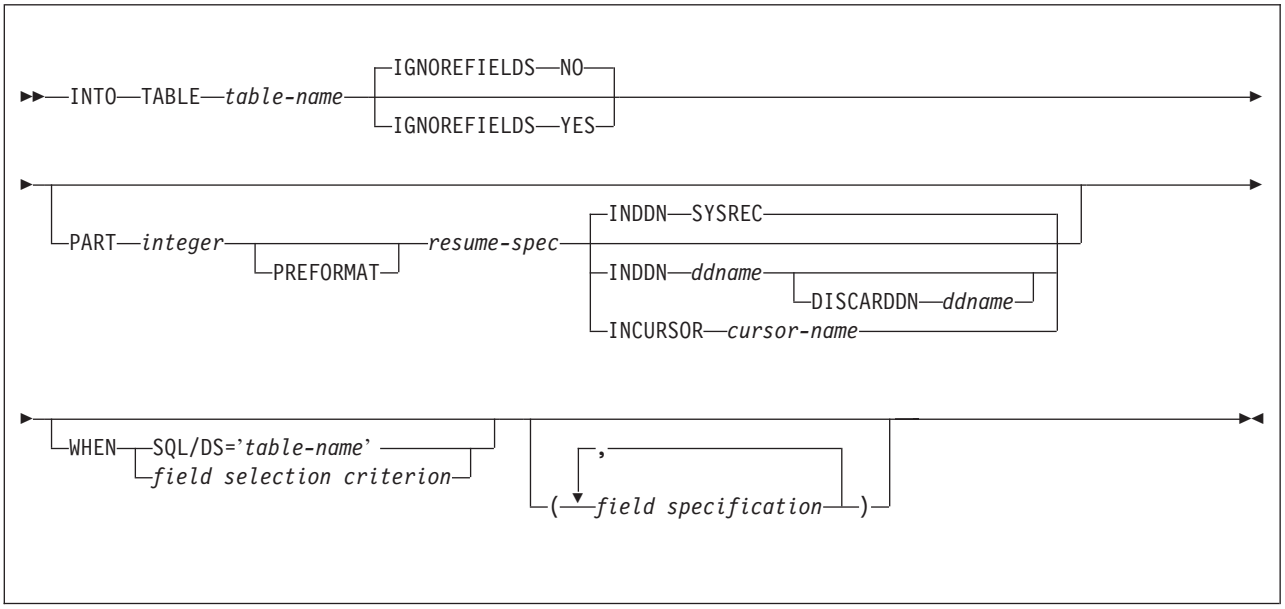
More than one table or partition for each table space can be loaded with a single invocation of the LOAD utility. At least one INTO TABLE statement is required for each table that is to be loaded. Each INTO TABLE statement:

- Identifies the table that is to be loaded
- Describes fields within the input record
- Defines the format of the input data set

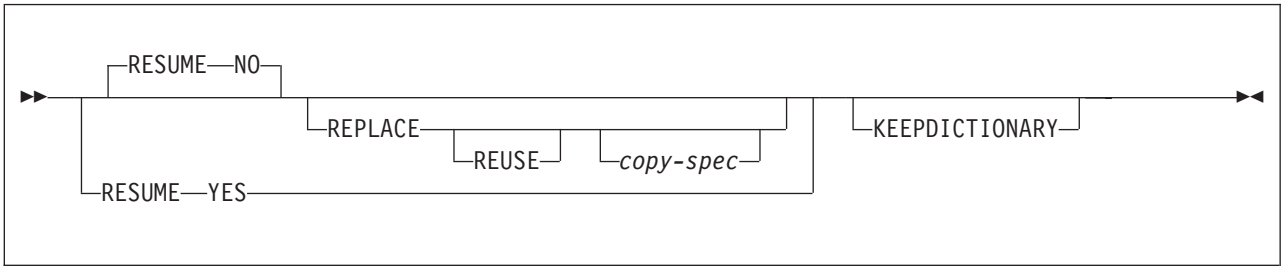
All tables that are specified by INTO TABLE statements must belong to the same table space.

If the data is already in UNLOAD or SQL/DS format, and FORMAT UNLOAD or FORMAT SQL/DS is used on the LOAD statement, no field specifications are allowed.

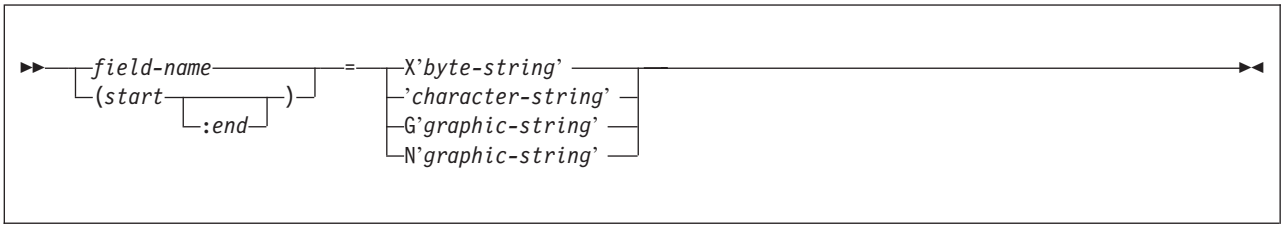
INTO-TABLE-spec:



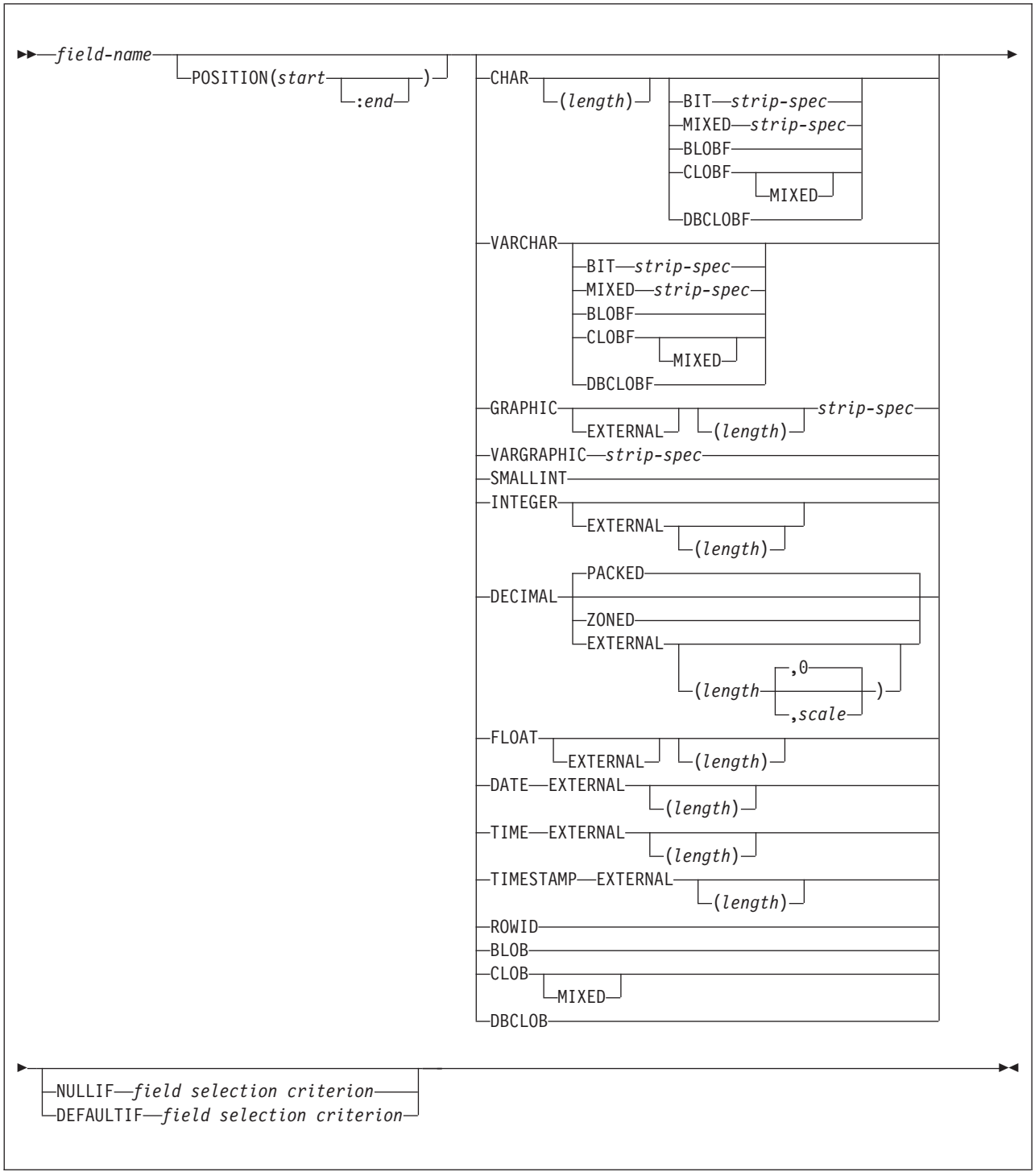
resume-spec:



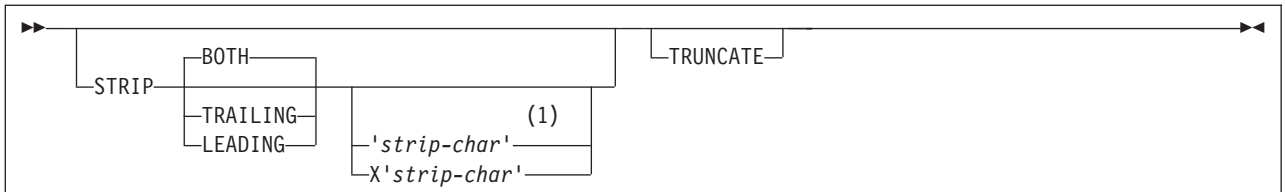
field selection criterion:



field specification:



strip spec:



Notes:

1 If you specify GRAPHIC or VARGRAPHIC, you cannot specify 'strip-char'. You can specify only X'strip-char'.

Option descriptions for INTO TABLE

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

table-name
Specifies the name of a table that is to be loaded. The table must be described in the catalog and must not be a catalog table or a system-maintained materialized query table.

If the table name is not qualified by an authorization ID, the authorization ID of the invoker of the utility job step is used as the qualifier of the table name. Enclose the table name in quotation marks if the name contains a blank.

Data from every LOAD record in the data set is loaded into the specified table unless:

- A WHEN clause is used, and the data does not match the field selection criterion.
- The FORMAT UNLOAD option is used on the LOAD statement, and the data comes from a table that is not specified in an INTO TABLE statement.
- A certain partition is specified, and the data does not belong to that partition.
- Data conversion errors occur.
- Any errors occur that are not generated by data conversion.

IGNOREFIELDS
Indicates whether LOAD is to skip fields in the input data set that do not correspond to columns in the target table. Examples of fields that do not correspond to table columns are the DSN_NULL_IND_*nnnnn*, DSN_ROWID, and DSN_IDENTITY fields that are generated by the REORG utility.

NO
Specifies that the LOAD process is not to skip any fields. The **default** is **NO**.

YES
Specifies that LOAD is to skip fields in the input data set that do not correspond to columns in the target table.

Specifying YES can be useful if each input record contains a variable-length field, followed by some variable-length data that you do not want to load and then some data that you want to load.

Because of the variable-length field, you cannot use the POSITION keyword to skip over the variable-length data that you do not want to load. By specifying IGNOREFIELDS, you can give a field specification for the variable-length data that you do not want to load; and by giving it a name that is not one of the table column names, LOAD skips the field without loading it.

Use this option with care, because it also causes fields to be skipped if you intend to load a column but have misspelled the name.

PART *integer*
Specifies that data is to be loaded into a partition of a partitioned table space. This option is valid only for partitioned table spaces.

| *integer* is the number of the partition into which records are to be loaded.
| The same partition number cannot be specified more than once if partition
| parallelism has been requested. Any data that is outside the range of the
| specified partition is not loaded. The maximum is 4096.

LOAD INTO PART *integer* is not allowed if an identity column is part of
the partitioning index, unless IDENTITYOVERRRIDE is specified for the
identity column GENERATED ALWAYS.

PREFORMAT
Specifies that the remaining pages are to be preformatted up to the high-allocated RBA in the partition and its corresponding partitioning index space. The preformatting occurs after the data is loaded and the indexes are built.

RESUME
Specifies whether records are to be loaded into an empty or non-empty partition. For nonsegmented table spaces, space is not reused for rows that have been marked as deleted or by rows of dropped tables is not reused. If the RESUME option is specified at the table space level, the RESUME option is not allowed in the PART clause.

If you want the RESUME option to apply to the entire table space, use the LOAD RESUME option. If you want the RESUME option to apply to a particular partition, specify it by using PART *integer* RESUME.

NO
Loads records into an empty partition. If the partition is not empty, and you have not used REPLACE, a message is issued, and the utility job step terminates with a job step condition code of 8.

For nonsegmented table spaces that contains deleted rows or rows of dropped tables, using the REPLACE keyword provides increased efficiency.

The **default** is **NO**.

YES
Loads records into a non-empty partition. If the partition is empty, a warning message is issued, but the partition is loaded.

REPLACE
Indicates that you want to replace only the contents of the partition that is cited by the PART option, rather than the entire table space.

You cannot use LOAD REPLACE with the PART *integer* REPLACE option of INTO TABLE. If you specify the REPLACE option, you must either replace an entire table space, using LOAD REPLACE, or a single partition,

using the PART *integer* REPLACE option of INTO TABLE. You can, however, use PART *integer* REPLACE with LOAD RESUME YES.

REUSE
Specifies, when used with the REPLACE option, that LOAD should logically reset and reuse DB2-managed data sets without deleting and redefining them. If you do not specify REUSE, DB2 deletes and redefines DB2-managed data sets to reset them.

If you specify REUSE with REPLACE on the PART specification (and not for LOAD at the table space level), only the specified partitions are logically reset. If you specify REUSE for the table space and REPLACE for the partition, data sets for the replaced parts are logically reset.

KEEPDICTIONARY
Specifies that the LOAD utility is not to build a new dictionary. LOAD retains the current dictionary and uses it for compressing the input data. This option eliminates the cost that is associated with building a new dictionary.

This keyword is valid only if a dictionary exists and the partition that is being loaded has the COMPRESS YES attribute.

If the partition has the COMPRESS YES attribute, but no dictionary exists, one is built and an error message is issued.

INDDN *ddname*
Specifies the data definition (DD) statement or template that identifies the input data set for the partition. The record format for the input data set must be fixed or variable. The data set must be readable by the basic sequential access method (BSAM).

The *ddname* is the name of the input data set. The default is **SYSREC**. INDDN can be a template name.

When loading LOB data, this input data set should include the names of
the files that contain the LOB column values. Each file can be either a PDS,
PDSE member, or separate HFS file.

If you specify INDDN, with or without DISCARDDN, in one INTO TABLE PART specification and you supply more than one INTO TABLE PART clause, you must specify INDDN in all INTO TABLE PART specifications.

Specifying INDDN at the partition level and supplying multiple PART clauses, each with their own INDDN, enables load partition parallelism, which can significantly improve performance. Loading all partitions in a single job with load partition parallelism is recommended instead of concurrent separate jobs whenever one or more nonpartitioned secondary indexes are on the table space.

The field specifications apply separately to each input file. Therefore, if multiple INTO TABLE PART INDDN clauses are used, field specifications are required on each one.

DISCARDDN *ddname*
Specifies the DD statement for a discard data set for the partition. The discard data set holds copies of records that are not loaded (for example, if they contain conversion errors). The discard data set also holds copies of records that were loaded and then removed (due to unique index errors, or referential or check constraint violations).

Flag input records for discarding during the RELOAD, INDEXVAL, and ENFORCE phases. However, the utility does not write the discard data set until the DISCARD phase when the utility copies the flagged records from the input data set to the discard data set.

The discard data set must be a sequential data set, and it must be write-accessible by BSAM, with the same record format, record length, and block size as the input data set.

The *ddname* is the name of the discard data set. DISCARDDN can be a template name.

If you omit the DISCARDDN option, LOAD does not save discarded records.

INCURSOR *cursor-name*

Specifies the cursor for the input data set. You must declare the cursor before it is used by the LOAD utility. Use the EXEC SQL utility control statement to define the cursor. You cannot load data into the same table on which you defined the cursor.

The specified cursor can be used as part of the DB2 UDB family cross loader function, which enables you to load data from any DRDA-compliant remote server. For more information about using the cross loader function, see "Loading data by using the cross-loader function" on page 251.

cursor-name is the cursor name. Cursor names that are specified with the LOAD utility cannot be longer than eight characters.

You cannot use the INCURSOR option with the following options:
- SHRLEVEL CHANGE
- NOSUBS
- FORMAT UNLOAD
- FORMAT SQL/DS
- CONTINUEIF
- WHEN.

In addition, you cannot specify field specifications with the INCURSOR option.

WHEN

Indicates which records in the input data set are to be loaded. If no WHEN clause is specified (and if FORMAT UNLOAD was not used in the LOAD statement), **all** records in the input data set are loaded into the specified tables or partitions. (Data that is beyond the range of the specified partition is not loaded.)

The option following WHEN describes a condition; input records that satisfy the condition are loaded. Input records that do not satisfy any WHEN clause of any INTO TABLE statement are written to the discard data set, if one is being used.

Data in the input record can be in ASCII or Unicode, but LOAD always interprets character constants that are specified in the utility control statement as EBCDIC. To use WHEN where the ASCII or UNICODE option is specified, code the condition by using the hexadecimal form, not the character string form. For example, use `(1:1)=X'31'` rather than `(1:1)='1'`. As an alternative, you can code the statement in UTF-8. See "Unicode character strings" on page 17for more information about hex notation and UTF-8.

SQL/DS='*table-name***'**
Is valid only when the FORMAT SQL/DS option is used on the LOAD statement.

table-name is the name of a table that has been unloaded into the unload data set. The table name after INTO TABLE tells which DB2 table the SQL/DS table is loaded into. Enclose the table name in quotation marks if the name contains a blank.

If no WHEN clause is specified, input records from every SQL/DS table are loaded into the table that is specified after INTO TABLE.

field-selection-criterion
Describes a field and a character constant. Only those records in which the field contains the specified constant are to be loaded into the table that is specified after INTO TABLE.

A field in a selection criterion must:

- Contain a character or graphic string. No data type conversions are performed when the contents of the field in the input record are compared to a string constant.
- Start at the same byte offset in each assembled input record. If any record contains varying-length strings, which are stored with length fields, that **precede** the selection field, they must be padded so that the start of the selection field is always at the same offset.

The field and the constant do not need to be the same length. If they are not, the shorter of the two is padded before a comparison is made. Character and graphic strings are padded with blanks. Hexadecimal strings are padded with zeros.

field-name
Specifies the name of a field that is defined by a *field-specification*. If *field-name* is used, the start and end positions of the field are given by the POSITION option of the field specification.

(start:end)
Identifies column numbers in the assembled load record; the first column of the record is column 1. The two numbers indicate the starting and ending columns of a selection field in the load record.

If *:end* is not used, the field is assumed to have the same length as the constant.

X'*byte-string***'**
Identifies the constant as a string of hexadecimal characters. For example, the following WHEN clause specifies that a record is to be loaded if it has the value X'FFFF' in columns 33 through 34.

```
WHEN (33:34) = X'FFFF'
```

'*character-string***'**
Identifies the constant as a string of characters. For example, the following WHEN clause specifies that a record is to be loaded if the field DEPTNO has the value D11.

```
WHEN DEPTNO = 'D11'
```

G'*graphic-string***'**
Identifies the constant as a string of double-byte characters. For example, the following WHEN clause specifies that a record is to be loaded if it has the specified value in columns 33 through 36.

```
WHEN (33:36) = G'<**>'
```

In this example, < is the shift-out character,* is a double-byte character, and > is the shift-in character.

If the first or last byte of the input data is a shift-out character, it is ignored in the comparison. Specify G as an uppercase character.

N'*graphic-string***'**
Identifies the constant as a string of double-byte characters. N and G are synonymous for specifying graphic string constants. Specify N as an uppercase character.

(*field-specification***, ...)**
Describes the location, format, and null value identifier of the data that is to be loaded.

If no field specifications are used:

- The fields in the input records are assumed to be in the same order as in the DB2 table.
- The formats are set by the FORMAT option on the LOAD statement, if that option is used.
- Fixed strings in the input are assumed to be of fixed maximum length. VARCHAR and VARGRAPHIC fields must contain a valid 2-byte binary length field preceding the data; no intervening gaps are allowed between the VARCHAR or VARGRAPHIC fields and the LOB that follow.
- ROWID fields are varying length, and must contain a valid 2-byte binary length field preceding the data; no intervening gaps are allowed between ROWID fields and the fields that follow.
- LOB fields are varying length, and require a valid 4-byte binary length field preceding the data; no intervening gaps are allowed between them and the LOB fields that follow.
- Numeric data is assumed to be in the appropriate internal DB2 number representation.
- The NULLIF or DEFAULTIF options cannot be used.

If any field specification is used for an input table, a field specification must exist for each field of the table that does not have a default value. Any field in the table with no corresponding field specification is loaded with its default value.

If any column in the output table does not have a field specification and is defined as NOT NULL, with no default, the utility job step is terminated.

Identity columns can appear in the field specification only if they were defined with the GENERATED BY DEFAULT attribute.

field-name
Specifies the name of a field, which can be a name of your choice. If the field is to be loaded, the name must be the name of a column in the table that is named after INTO TABLE unless IGNOREFIELDS is specified. You can use the field name as a vehicle to specify the range of incoming data. See “Example 4: Loading data of different data types” on page 275 for an example of loading selected records into an empty table space.

The starting location of the field is given by the POSITION option. If POSITION is not used, the starting location is one column after the end of the previous field.

LOAD determines the length of the field in one of the following ways, in the order listed:

1. If the field has data type VARCHAR, VARGRAPHIC, or ROWID, the length is assumed to be contained in a 2-byte binary field that precedes the data. For VARCHAR fields, the length is in bytes; for VARGRAPHIC fields, the length field identifies the number of double-byte characters.

   If the field has data type CLOB, BLOB, or DBCLOB, the length is assumed to be contained in a 4-byte binary field that precedes the data. For BLOB and CLOB fields, the length is in bytes; for DBCLOB fields, the length field identifies the number of double-byte characters.
2. If *:end* is used in the POSITION option, the length is calculated from *start* and *end*. In that case, any length attribute after the CHAR, GRAPHIC, INTEGER, DECIMAL, or FLOAT specifications is ignored.
3. The length attribute on the CHAR, GRAPHIC, INTEGER, DECIMAL, or FLOAT specifications is used as the length.
4. The length is taken from the DB2 field description in the table definition, or it is assigned a default value according to the data type. For DATE and TIME fields, the length is defined during installation. For variable-length fields, the length is defined from the column in the DB2 table definition, excluding the null indicator byte, if it is present. Table 28 shows the default length, in bytes, for each data type.

Table 28. Default length of each data type (in bytes)

Data type	Default length in bytes
BLOB	Varying
CHARACTER	Length that is used in column definition
CLOB	Varying
DATE	10 (or installation default)
DBCLOB	Varying
DECIMAL EXTERNAL	Decimal precision for output columns that are decimal, otherwise the length that is used in column definition
DECIMAL PACKED	Length that is used in column definition
DECIMAL ZONED	Decimal precision for output columns that are decimal, otherwise the length that is used in column definition
FLOAT (single precision)	4
FLOAT (double precision)	8
GRAPHIC	2 multiplied by (length that is used in column definition)
INTEGER	4
MIXED	Mixed DBCS data
ROWID	Varying
SMALLINT	2
TIME	8 (or installation default)
TIMESTAMP	26
VARCHAR	Varying

Table 28. Default length of each data type (in bytes) (continued)

Data type	Default length in bytes
VARGRAPHIC	Varying

If a data type is not given for a field, its data type is assumed to be the same as that of the column into which it is loaded, as given in the DB2 table definition.

POSITION(*start:end***)**
Indicates where a field is in the assembled load record.

start and *end* are the locations of the first and last columns of the field; the first column of the record is column 1. The option can be omitted.

Column locations can be specified as:
- An integer *n*, meaning an actual column number
- *, meaning one column after the end of the previous field
- *+*n*, where *n* is an integer, meaning *n* columns after the location that is specified by *

Do not enclose the entire POSITION option specification in parentheses; enclose only the *start:end* description in parentheses. Valid and invalid specifications are shown in Table 29.

Table 29. Example of valid and invalid POSITION specifications

Valid	Invalid
`POSITION (10:20)`	`(POSITION (10:20))`

Data types in a field specification: The data type of the field can be specified by any of the keywords that follow. Except for graphic fields, *length* is the length in bytes of the input field.

All numbers that are designated EXTERNAL are in the same format in the input records.

CHAR(*length***)**
Specifies a fixed-length character string. If you do not specify *length*, the length of the string is determined from the POSITION specification. If you do not specify *length* or POSITION, LOAD uses the default length for CHAR, which is determined from the length of the column in the table. See Table 28 on page 222 for more information on the default length for CHAR. You can also specify CHARACTER and CHARACTER(*length*).

| **BIT**
| Specifies that the input field contains BIT data. If BIT is specified, LOAD bypasses any CCSID conversions for the input data. If the target column has the BIT data type attribute, LOAD bypasses any code page translation for the input data.

MIXED
Specifies that the input field contains mixed SBCS and DBCS data. If MIXED is specified, any required CCSID conversions use the mixed CCSID for the input data. If MIXED is not specified, any such conversions use the SBCS CCSID for the input data.

BLOBF
# Indicates that the input field contains the name of a file from which a
BLOB is to be loaded into the specified table without CCSID conversion.

CLOBF
Indicates that the input field contains the name of a file from which a CLOB is to be loaded into the specified table with any required CCSID conversion.

DBCLOBF
Indicates that the input field contains the name of a file from which a DBCLOB is to be loaded into the specified table with any required CCSID conversion.

STRIP
Specifies that LOAD is to remove blanks (the default) or the specified characters from the beginning, the end, or both ends of the data. LOAD pads the CHAR field, so that it fills the rest of the column.

LOAD applies the strip operation before performing any character code conversion or padding.

The effect of the STRIP option is the same as the SQL STRIP scalar function. For details, see Chapter 5 of *DB2 SQL Reference*.

BOTH
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the beginning and end of the data. The **default** is **BOTH**.

TRAILING
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the end of the data.

LEADING
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the beginning of the data.

'strip-char'
Specifies a single-byte or double-byte character that LOAD is to strip from the data.

Specify this character value in EBCDIC. Depending on the input encoding scheme, LOAD applies SBCS CCSID conversion to the *strip-char* value before it is used in the strip operation.

If the subtype of the column to be loaded is BIT or you want to specify a *strip-char* value in an encoding scheme other than EBCDIC, use the hexadecimal form (*X'strip-char'*). LOAD does not perform any CCSID conversion if the hexadecimal form is used.

X'strip-char'
Specifies in hexadecimal form a single-byte or double-byte character that LOAD is to strip from the data. For single-byte characters, specify this value in the form X'hh', where *hh* is two hexadecimal characters. For double-byte characters, specify this value in the form X'hhhh', where *hhhh* is four hexadecimal characters.

Use the hexadecimal form to specify a character in an encoding scheme other than EBCDIC. When you specify the character value in hexadecimal form, LOAD does not perform any CCSID conversion.

If you specify a strip character in the hexadecimal format, you must specify the character in the input encoding scheme.

TRUNCATE
Indicates that LOAD is to truncate the input character string from the right

if the string does not fit in the target column. LOAD performs the truncation operation after any CCSID translation.

If the input data is BIT data, LOAD truncates the data at a byte boundary. If the input data is SBCS or MIXED data, LOAD truncates the data at a character boundary. (Double-byte characters are not split.) If a MIXED field is truncated to fit a column, the truncated string can be shorter than the specified column size. In this case, blanks in the output CCSID are padded to the right. If MIXED data is in EBCDIC, truncation preserves the SO (shift-out) and SI (shift-in) characters around a DBCS string.

VARCHAR
Specifies a character field of varying length. The length in bytes must be specified in a 2-byte binary field preceding the data. (The length does not include the 2-byte field itself.) The length field must start in the column that is specified as *start* in the POSITION option. If *:end* is used, it is ignored.

BIT
Specifies that the input field contains BIT data. If BIT is specified, LOAD bypasses any CCSID conversions for the input data. If the target column has the BIT data type attribute, LOAD bypasses any code page translation for the input data.

MIXED
Specifies that the input field contains mixed DBCS data. If MIXED is specified, any required CCSID conversions use the mixed CCSID for the input data. If MIXED is not specified, any such conversions use the SBCS CCSID for the input data.

BLOBF
Indicates that the input field contains the name of a file from which a BLOB is to be loaded into the specified table without CCSID conversion. The file name must be in the SBCS CCSID of the specified encoding scheme.

CLOBF
Indicates that the input field contains the name of a file from which a CLOB is to be loaded into the specified table with any required CCSID conversion. The file name must be in the SBCS CCSID of the specified encoding scheme.

DBCLOBF
Indicates that the input field contains the name of a file from which a DBCLOB is to be loaded into the specified table with any required CCSID conversion. The file name must be in the SBCS CCSID of the specified encoding scheme.

STRIP
Specifies that LOAD is to remove blanks (the default) or the specified characters from the beginning, the end, or both ends of the data. LOAD adjusts the VARCHAR length field to the length of the stripped data.

LOAD applies the strip operation before performing any character code conversion or padding.

The effect of the STRIP option is the same as the SQL STRIP scalar function. For details, see Chapter 5 of *DB2 SQL Reference*.

BOTH
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the beginning and end of the data. The **default** is **BOTH**.

TRAILING
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the end of the data.

LEADING
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the beginning of the data.

'strip-char'
Specifies a single-byte or double-byte character that LOAD is to strip from the data.

Specify this character value in EBCDIC. Depending on the input encoding scheme, LOAD applies SBCS CCSID conversion to the *strip-char* value before it is used in the strip operation.

If the subtype of the column to be loaded is BIT or you want to specify a *strip-char* value in an encoding scheme other than EBCDIC, use the hexadecimal form (*X'strip-char'*). LOAD does not perform any CCSID conversion if the hexadecimal form is used.

X'strip-char'
Specifies in hexadecimal form a single-byte or double-byte character that LOAD is to strip from the data. For single-byte characters, specify this value in the form X'hh', where *hh* is two hexadecimal characters. For double-byte characters, specify this value in the form X'hhhh', where *hhhh* is four hexadecimal characters.

Use the hexadecimal form to specify a character in an encoding scheme other than EBCDIC. When you specify the character value in hexadecimal form, LOAD does not perform any CCSID conversion.

If you specify a strip character in the hexadecimal format, you must specify the character in the input encoding scheme.

TRUNCATE
Indicates that LOAD is to truncate the input character string from the right if the string does not fit in the target column. LOAD performs the truncation operation after any CCSID translation.

If the input data is BIT data, LOAD truncates the data at a byte boundary. If the input data is character type data, LOAD truncates the data at a character boundary. If a mixed-character type data is truncated to fit a column of fixed size, the truncated string can be shorter than the specified column size. In this case, blanks in the output CCSID are padded to the right.

GRAPHIC(*length***)**
Specifies a fixed-length graphic type. You can specify both *start* and *end* for the field specification.

If you use GRAPHIC, the input data must not contain shift characters. *start* and *end* must indicate the starting and ending positions of the data itself.

length is the number of double-byte characters. The length of the field in bytes is twice the value of *length*. If you do not specify *length*, the number of double-byte characters is determined from the POSITION specification. If you

do not specify *length* or POSITION, LOAD uses the default length for GRAPHIC, which is determined from the length of the column in the table. See Table 28 on page 222 for more information on the default length for GRAPHIC.

For example, let *** represent three double-byte characters. Then, to describe ***, specify either `POS(1:6) GRAPHIC` or `POS(1) GRAPHIC(3)`. A GRAPHIC field that is described in this way cannot be specified in a field selection criterion.

STRIP
Specifies that LOAD is to remove blanks (the default) or the specified characters from the beginning, the end, or both ends of the data.

LOAD applies the strip operation before performing any character code conversion or padding.

The effect of the STRIP option is the same as the SQL STRIP scalar function. For details, see Chapter 5 of *DB2 SQL Reference*.

BOTH
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the beginning and end of the data. The **default** is **BOTH**.

TRAILING
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the end of the data.

LEADING
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the beginning of the data.

X'strip-char'
Specifies the hexadecimal form of the double-byte character that LOAD is to strip from the data. Specify this value in the form X'hhhh', where *hhhh* is four hexadecimal characters.

You must specify the character in the input encoding scheme.

TRUNCATE
Indicates that LOAD is to truncate the input character string from the right if the string does not fit in the target column. LOAD performs the truncation operation after any CCSID translation.

LOAD truncates the data at a character boundary. Double-byte characters are not split.

GRAPHIC EXTERNAL(*length***)**
Specifies a fixed-length field of the graphic type with the external format. You can specify both *start* and *end* for the field specification.

If you use GRAPHIC EXTERNAL, the input data must contain a shift-out character in the starting position, and a shift-in character in the ending position. Other than the shift characters, this field must have an even number of bytes. The first byte of any pair must not be a shift character.

length is the number of double-byte characters. *length* for GRAPHIC EXTERNAL does not include the number of bytes that are represented by shift characters. The length of the field in bytes is twice the value of *length*. If you do not specify *length*, the number of double-byte characters is determined from the POSITION specification. If you do not specify *length* or POSITION, LOAD uses the default length for GRAPHIC, which is determined from the length of the column in the table. See Table 28 on page 222 for more information on the default length for GRAPHIC.

For example, let *** represent three double-byte characters, and let < and > represent shift-out and shift-in characters. Then, to describe <***>, specify either `POS(1:8) GRAPHIC EXTERNAL` or `POS(1) GRAPHIC EXTERNAL(3)`.

STRIP
Specifies that LOAD is to remove blanks (the default) or the specified characters from the beginning, the end, or both ends of the data.

LOAD applies the strip operation before performing any character code conversion or padding.

The effect of the STRIP option is the same as the SQL STRIP scalar function. For details, see Chapter 5 of *DB2 SQL Reference*.

BOTH
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the beginning and end of the data. The **default** is **BOTH**.

TRAILING
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the end of the data.

LEADING
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the beginning of the data.

X'strip-char'
Specifies the hexadecimal form of the double-byte character that LOAD is to strip from the data. Specify this value in the form X'hhhh', where *hhhh* is four hexadecimal characters.

You must specify the character in the input encoding scheme.

TRUNCATE
Indicates that LOAD is to truncate the input character string from the right if the string does not fit in the target column. LOAD performs the truncation operation after any CCSID translation.

LOAD truncates the data at a character boundary. Double-byte characters are not split.

VARGRAPHIC
Identifies a graphic field of varying length. The length, in double-byte characters, must be specified in a 2-byte binary field preceding the data. (The length does not include the 2-byte field itself.) The length field must start in the column that is specified as *start* in the POSITION option. *:end*, if used, is ignored.

VARGRAPHIC input data must not contain shift characters.

STRIP
Specifies that LOAD is to remove blanks (the default) or the specified characters from the beginning, the end, or both ends of the data. LOAD adjusts the VARGRAPHIC length field to the length of the stripped data (the number of DBCS characters).

LOAD applies the strip operation before performing any character code conversion or padding.

The effect of the STRIP option is the same as the SQL STRIP scalar function. For details, see Chapter 5 of *DB2 SQL Reference*.

BOTH
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the beginning and end of the data. The **default** is **BOTH**.

TRAILING
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the end of the data.

LEADING
Indicates that LOAD is to remove occurrences of blank or the specified strip character from the beginning of the data.

X'strip-char'
Specifies the hexadecimal form of the double-byte character that LOAD is to strip from the data. Specify this value in the form X'hhhh', where *hhhh* is four hexadecimal characters.

You must specify the character in the input encoding scheme.

TRUNCATE
Indicates that LOAD is to truncate the input character string from the right if the string does not fit in the target column. LOAD performs the truncation operation after any CCSID translation.

LOAD truncates the data at a character boundary. Double-byte characters are not split.

SMALLINT
Specifies a 2-byte binary number. Negative numbers are in two's complement notation.

INTEGER
Specifies a 4-byte binary number. Negative numbers are in two's complement notation. You can also specify INT.

INTEGER EXTERNAL(*length*)
A string of characters that represent a number. The format is that of an SQL numeric constant, as described in Chapter 2 of *DB2 SQL Reference*. If you do not specify *length*, the length of the string is determined from the POSITION specification. If you do not specify *length* or POSITION, LOAD uses the default length for INTEGER, which is 4 bytes. See Table 28 on page 222 for more information on the default length for INTEGER. You can also specify INT EXTERNAL.

DECIMAL PACKED
Specifies a number of the form *ddd...ds*, where *d* is a decimal digit that is represented by four bits, and *s* is a 4-bit sign value. The plus sign (+) is represented by A, C, E, or F, and the minus sign (-) is represented by B or D. The maximum number of *d*s is the same as the maximum number of digits that are allowed in the SQL definition. You can also specify DECIMAL, DEC, or DEC PACKED.

DECIMAL ZONED
Specifies a number in the form *znznzn...z/sn*, where *z*, *n*, and *s* have the following values:

n A decimal digit represented by the right 4 bits of a byte (called the *numeric bits*)
z That digit's zone, represented by the left 4 bits
s The right-most byte of the decimal operand; *s* can be treated as a zone or as the sign value for that digit

The plus sign (+) is represented by A, C, E, or F, and the minus sign (-) is represented by B or D. The maximum number of *zn*s is the same as the maximum number of digits that are allowed in the SQL definition. You can also specify DEC ZONED.

DECIMAL EXTERNAL(*length,scale***)**
Specifies a string of characters that represent a number. The format is that of an SQL numeric constant, as described in Chapter 2 of *DB2 SQL Reference*.

length
Overall length of the input field, in bytes. If you do not specify *length*, the length of the input field is determined from the POSITION specification. If you do not specify *length* or POSITION, LOAD uses the default length for DECIMAL EXTERNAL, which is determined by using decimal precision. See Table 28 on page 222 for more information on the default length for DECIMAL EXTERNAL.

scale
Specifies the number of digits to the right of the decimal point. *scale* must be an integer greater than or equal to 0, and it can be greater than *length*. The **default** is **0**.

If *scale* is greater than *length*, or if the number of provided digits is less than the *specified scale*, the input number is padded on the left with zeros until the decimal point position is reached. If *scale* is greater than the target *scale*, the source scale locates the implied decimal position. All fractional digits greater than the target scale are truncated. If scale is specified and the target column has a data type of small integer or integer, the decimal portion of the input number is ignored. If a decimal point is present, its position overrides the field specification of *scale*.

FLOAT(*length***)**
Specifies either a 64-bit floating-point number or a 32-bit floating-point number. If *length* is between 1 and 21 inclusive, the number is 32 bits in the s390 (HFP) format:

Bit 0 Represents a sign (0 for plus and 1 for minus)

Bits 1-7 Represent an exponent

Bits 8-31 Represent a mantissa

If *length* is between 1 and 24 inclusive, the number is 32 bits in the IEEE (BFP) format:

Bit 0 Represents a sign (0 for plus and 1 for minus)

Bits 1-8 Represent an exponent

Bits 9-31 Represent a mantissa

If *length* is not specified, or is between 22 and 53 inclusive, the number is 64 bits in the s390 (HFP) format:

Bit 0 Represents a sign (0 for plus and 1 for minus)

Bits 1-7 Represent an exponent

Bits 8-63 Represent a mantissa.

If *length* is not specified, or is between 25 and 53 inclusive, the number is 64 bits in the IEEE (BFP) format:

Bit 0 Represents a sign (0 for “plus”, and 1 for “minus”)

Bits 1-11 Represent an exponent

Bits 12-63 Represent a mantissa.

You can also specify REAL for single-precision floating-point numbers and DOUBLE PRECISION for double-precision floating-point numbers.

FLOAT EXTERNAL(*length*)
Specifies a string of characters that represent a number. The format is that of an SQL floating-point constant, as described in Chapter 2 of *DB2 SQL Reference*.

A specification of FLOAT(IEEE) or FLOAT(S390) does not apply for this format (string of characters) of floating-point numbers.

If you do not specify *length*, the length of the string is determined from the POSITION specification. If you do not specify *length* or POSITION, LOAD uses the default length for FLOAT, which is 4 bytes for single precision and 8 bytes for double precision. See Table 28 on page 222 for more information on the default length for FLOAT.

DATE EXTERNAL(*length*)
Specifies a character string representation of a date. The length, if unspecified, is the specified length on the LOCAL DATA LENGTH install option, or, if none was provided, the default is 10 bytes. If you specify a length, it must be within the range of 8 to 254 bytes.

Dates can be in any of the following formats. You can omit leading zeros for month and day. You can include trailing blanks, but no leading blanks are allowed.
- *dd.mm.yyyy*
- *mm/dd/yyyy*
- *yyyy-mm-dd*
- Any local format that your site defined at the time DB2 was installed

TIME EXTERNAL(*length*)
Specifies a character string representation of a time. The length, if unspecified, is the specified length on the LOCAL TIME LENGTH install option, or, if none was provided, the default is 8 bytes. If you specify a length, it must be within the range of 4 to 254 bytes.

Times can be in any of the following formats:
- *hh.mm.ss*
- *hh:mm* AM
- *hh:mm* PM
- *hh:mm:ss*
- Any local format that your site defined at the time DB2 was installed

You can omit the *mm* portion of the *hh:mm* AM and *hh:mm* PM formats if *mm* is equal to 00. For example, 5 PM is a valid time, and can be used instead of 5:00 PM.

TIMESTAMP EXTERNAL(*length*)
Specifies a character string representation of a time. The default for *length* is 26 bytes. If you specify a length, it must be within the range of 19 to 26 bytes.

Timestamps can be in any of the following formats. Note that *nnnnnn* represents the number of microseconds, and can be from 0 to 6 digits. You can omit leading zeros from the month, day, or hour parts of the timestamp; you can omit trailing zeros from the microseconds part of the timestamp.
- *yyyy-mm-dd-hh.mm.ss*
- *yyyy-mm-dd-hh.mm.ss.nnnnnn*

- *yyyy-mm-dd hh:mm:ss.nnnnnn*

See Chapter 2 of *DB2 SQL Reference* for more information about the timestamp data type.

ROWID
Specifies a row ID. The input data must be a valid value for a row ID; DB2 does not perform any conversions.

A field specification for a row ID column is not allowed if the row ID column was created with the GENERATED ALWAYS option.

If the row ID column is part of the partitioning key, LOAD INTO TABLE PART is not allowed; specify LOAD INTO TABLE instead.

BLOB
Specifies a BLOB field. You must specify the length in bytes in a 4-byte binary field that precedes the data. (The length does **not** include the 4-byte field itself.) The length field must start in the column that is specified as *start* in the POSITION option. If *:end* is used, it is ignored.

CLOB
Specifies a CLOB field. You must specify the length in bytes in a 4-byte binary field that precedes the data. (The length does **not** include the 4-byte field itself.) The length field must start in the column that is specified as *start* in the POSITION option. If *:end* is used, it is ignored.

MIXED
Specifies that the input field contains mixed SBCS and DBCS data. If MIXED is specified, any required CCSID conversions use the mixed CCSID for the input data; if MIXED is not specified, any such conversions use the SBCS CCSID for the input data.

DBCLOB
Specifies a DBCLOB field. You must specify the length in double-byte characters in a 4-byte binary field that precedes the data. (The length does **not** include the 4-byte field itself.) The length field must start in the column that is specified as *start* in the POSITION option. If *:end* is used, it is ignored.

DEFAULTIF *field-selection-criterion*
Describes a condition that causes the DB2 column to be loaded with its default value. You can write the *field-selection-criterion* with the same options as described under "field-selection-criterion" on page 220. If the contents of the DEFAULTIF field match the provided character constant, the field that is specified in *field-specification* is loaded with its default value.

If the DEFAULTIF field is defined by the name of a VARCHAR or VARGRAPHIC field, DB2 takes the length of the field from the 2-byte binary field that appears before the data portion of the VARCHAR or VARGRAPHIC field.

Data in the input record can be in ASCII or Unicode, but the utility interprets character constants that are specified in the utility control statement as EBCDIC or Unicode. If the control statement is in the same encoding scheme as the input data, you can code character constants in the control statement. Otherwise, if the control statement is not in the same encoding scheme as the input data, you must code the condition with hexadecimal constants. For example, if the input data is in EBCDIC and the control statement is in UTF-8, use `(1:1)=X'31'` in the condition rather than `(1:1)='1'`. See "Unicode character strings" on page 17for more information about hex notation and UTF-8.

You can use the DEFAULTIF attribute with the ROWID keyword. If the condition is met, the column is loaded with a value that DB2 generates.

NULLIF *field-selection-criterion*

Describes a condition that causes the DB2 column to be loaded with NULL. You can write the *field-selection-criterion* with the same options as described under "field-selection-criterion" on page 220. If the contents of the NULLIF field match the provided character constant, the field that is specified in *field-specification* is loaded with NULL.

If the NULLIF field is defined by the name of a VARCHAR or VARGRAPHIC field, DB2 takes the length of the field from the 2-byte binary field that appears before the data portion of the VARCHAR or VARGRAPHIC field.

To load a null value into a BLOBF, CLOBF, or DBCLOBF field, use a null input
file name.

| Data in the input record can be in ASCII or Unicode, but the utility interprets
| character constants that are specified in the utility control statement as EBCDIC
| or Unicode. If the control statement is in the same encoding scheme as the
| input data, you can code character constants in the control statement.
| Otherwise, if the control statement is not in the same encoding scheme as the
| input data, you must code the condition with hexadecimal constants. For
| example, if the input data is in EBCDIC and the control statement is in UTF-8,
| use `(1:1)=X'31'` in the condition rather than `(1:1)='1'`. See "Unicode
| character strings" on page 17for more information about hex notation and
| UTF-8.

The fact that a field in the output table is loaded with NULL does not change the format or function of the corresponding field in the input record. The input field can still be used in a field selection criterion. For example, assume that a LOAD statement has the following field specification:

```
(FIELD1 POSITION(*) CHAR(4)
 FIELD2 POSITION(*) CHAR(3) NULLIF(FIELD1='SKIP')
 FIELD3 POSITION(*) CHAR(5))
```

Assume also that LOAD is to process the following source record:

```
SKIP   FLD03
```

In this example, the record is loaded as follows:

FIELD1
Has the value 'SKIP'.

FIELD2
Is NULL (not ' ' as in the source record).

FIELD3
Has the value 'FLD03'.

You cannot use the NULLIF parameter with the ROWID keyword because row ID columns cannot be null.

Field selection criterion

Describes a condition that causes the DB2 column to be loaded with NULL or with its default value.

Instructions for running LOAD

To run LOAD, you must:

1. Read "Before running LOAD" in this section.
2. Prepare the necessary data sets, as described in "Data sets that LOAD uses" on page 235.
3. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for LOAD, see "Sample LOAD control statements" on page 274.)
4. Prepare a utility control statement that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks" on page 239.
5. Check the compatibility table in "Concurrency and compatibility for LOAD" on page 267 if you want to run other jobs concurrently on the same target objects.
6. Plan for restart if the LOAD job doesn't complete, as described in "Terminating or restarting LOAD" on page 264.
7. Read "After running LOAD" on page 269 in this section.
8. Run LOAD by using one of the methods described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Before running LOAD

You cannot run the LOAD utility on the DSNDB01 or DSNDB06 databases, except to add rows to the following catalog tables:

- SYSSTRINGS
- MODESELECT
- LUMODES
- LULIST
- USERNAMES
- LUNAMES
- LOCATIONS
- IPNAMES

Preprocessing input data

No sorting of the data rows occurs during LOAD processing. Rows are loaded in the physical sequence in which they are found.

Recommendation: Sort your input records in clustering sequence before loading the data.

You should also:

- Ensure that no duplicate keys exist for unique indexes.
- Correct check constraint violations and referential constraint violations in the input data set.
- Ensure that any input data that is provided for a security label column is a valid security label. Security label columns are defined with the AS SECURITY LABEL clause. These columns are used for multilevel security with row-level granularity. For more information about multilevel security and security labels, see Part 3 of *DB2 Administration Guide*.

When loading data into a segmented table space, sort your data by table to ensure that the data is loaded in the best physical organization.

Loading data by using a cursor

Before you can load data by using a cursor, you need to bind the DSNUT810 package at each location from which you plan to load data. A local package for DSNUT810 is bound by installation job DSNTIJSG when you install or migrate to a new version of DB2 UDB for z/OS.

The following example statement binds the DSNUT810 package at a remote location:

```
BIND PACKAGE(location.DSNUT810) MEMBER(DSNUGSQL) -
       ACTION(REPLACE) ISOLATION(CS) ENCODING(EBCDIC) -
       VALIDATE(BIND) CURRENTDATA(NO) -
       LIBRARY('prefix.SDSNDBRM')
```

Data sets that LOAD uses

Table 30 lists the data sets that LOAD uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set, and any optional data sets that you want to use. Alternatively, you can use the TEMPLATE utility to dynamically allocate some of these data sets.

Table 30. Data sets that LOAD uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
STPRIN01	A data set that contains messages from DFSORT (usually, SYSOUT or DUMMY). This data set is used when frequency statistics are collected on DPSI's or when TABLESPACE TABLE COLGROUP FREQVAL is specified	No[1]
Input data set	The input data set that contains the data that is to be loaded. Specify its template or DD name with the INDDN option of the utility control statement. The default name is SYSREC. It must be a sequential data set that is readable by BSAM.	Yes[2]
Sort data sets	Two temporary work data sets for sort input and sort output. Specify their DD or template names with the WORKDDN option of the utility control statement. The default DD name for sort input is SYSUT1. The default DD name for sort output is SORTOUT.	Yes[3, 4]
Mapping data set	Work data set for mapping the identifier of a table row back to the input record that caused an error. Specify its template or DD name with the MAPDDN option of the utility control statement. The default DD name is SYSMAP.	Yes[3, 5]
UTPRINT	Contains messages from DFSORT (usually, SYSOUT or DUMMY).	No[6]

Table 30. Data sets that LOAD uses (continued)

Data set	Description	Required?
Discard data set	A work data set that contains copies of records that are not loaded. It must be a sequential data set that is readable by BSAM. Specify its DD or template name with the DISCARDDN option of the utility control statement. The default DD name is SYSDISC.	Yes [7]
Error data set	Work data set for error processing. Specify its DD or template name with the ERRDDN option of the utility control statement. The default DD or template name is SYSERR.	Yes
Copy data sets	One to four output data sets that contain image copy data sets. Specify their DD or template names with the COPYDDN and RECOVERYDDN options of the utility control statement.	No[8]
Sort work data sets	Temporary data sets for sort input and output when sorting keys. If index build parallelism is used, the DD names have the form SW*nn*WK*mm*. If index build parallelism is not used, the DD names have the form SORTWK*nn*. For more information about allocating these data sets, see "Building indexes in parallel for LOAD" on page 258.	Yes[9]
Sort work data sets	Temporary data sets for sort input and output when collecting inline statistics on at least one data-partitioned secondary index. The DD names have the form ST01WK*nn*.	No[1, 10, 11]

Notes:

1. Required when collecting inline statistics on at least one data-partitioned secondary index.
2. As an alternative to specifying an input data set, you can specify a cursor with the INCURSOR option. For more information about cursors, see "Loading data by using the cross-loader function" on page 251.
3. Required if referential constraints exist and ENFORCE(CONSTRAINTS) is specified (This option is the default).
4. Used for tables with indexes.
5. Required for discard processing when loading one or more tables that have unique indexes.
6. Required if a sort is done.
7. If you omit the DD statement for this data set, LOAD creates the data set with the same record format, record length, and block size as the input data set.
8. Required for inline copies.
9. Required if any indexes are to be built or if a sort is required for processing errors.
10. If the DYNALLOC parm of the SORT program is not turned on, you need to allocate the data set. Otherwise, DFSORT dynamically allocates the temporary data set.
11. It is recommended that you use dynamic allocation by specifying SORTDEVT in the utility statement because dynamic allocation reduces the maintenance required of the utility job JCL.

The following object is named in the utility control statement and does not require a DD statement in the JCL:

Table Table that is to be loaded. (If you want to load only one partition of a table, you must use the PART option in the control statement.)

Defining work data sets: Use the formulas and instructions in Table 31 to calculate the size of work data sets for LOAD. Each row in the table lists the DD name that is used to identify the data set and either formulas or instructions that you should use to determine the size of the data set. The key for the formulas is located at the bottom of the table.

Table 31. Size of work data sets for LOAD jobs

Work data set	Size
SORTOUT	*max(f,e)*
ST01WK*nn*	2 ×(*maximum record length* × *numcols* × (*count* + 2) × number of indexes)
SYSDISC	Same size as input data set
SYSERR	*e*
SYSMAP	• Simple table space for discard processing: *m* • Partitioned or segmented table space without discard processing: *max(m,e)*
SYSUT1	• Simple table space: *max(k,e)* • Partitioned or segmented table space: *max(k,e,m)* If you specify an estimate of the number of keys with the SORTKEYS option: *max(f,e)* for a simple table space *max(f,e,m)* for a partitioned or segmented table space

Note:

variable
meaning

k	Key calculation
f	Foreign key calculation
m	Map calculation
e	Error calculation
max()	Maximum value of the specified calculations
numcols	Number of key columns to concatenate when you collect frequent values from the specified index
count	Number of frequent values that DB2 is to collect

maximum record length
Maximum record length of the SYSCOLDISTSTATS record that is processed when collecting frequency statistics (You can obtain this value from the RECLENGTH column in SYSTABLES.)

Calculating the key: k

If a mix of data-partitioned secondary indexes and nonpartitioned indexes exists on the table that is being loaded or a foreign key exists that is exactly indexed by a data-partitioned secondary index, use this formula:

max(longest index key + 15, longest foreign key + 15) × (number of extracted keys).

Otherwise, use this formula:

max(longest index key + 13, longest foreign key + 13) × (number of extracted keys).

For nonpadded indexes, the length of the longest key means the maximum possible length of a key with all varying-length columns padded to their maximum lengths, plus 2 bytes for each varying-length column.

Calculating the number of extracted keys:

1. Count 1 for each index.
2. Count 1 for each foreign key that is not exactly indexed (that is, where foreign key and index definitions do not correspond identically).
3. For each foreign key that is exactly indexed (that is, where foreign key and index definitions correspond identically):
   a. Count 0 for the first relationship in which the foreign key participates if the index is not a data-partitioned secondary index. Count 1 if the index is a data-partitioned secondary index.
   b. Count 1 for subsequent relationships in which the foreign key participates (if any).
4. Multiply count by the number of rows that are to be loaded.

Calculating the foreign key: f

If a mix of data-partitioned secondary indexes and nonpartitioned indexes exists on the table that is being loaded or a foreign key exists that is exactly indexed by a data-partitioned secondary index, use this formula:

max(longest foreign key + 15) × (number of extracted keys)

Otherwise, use this formula:

max(longest foreign key + 13) × (number of extracted keys)

Calculating the map: m

The data set must be large enough to accommodate one map entry (length = 21 bytes) per table row that is produced by the LOAD job.

Calculating the error: e

The data set must be large enough to accommodate one error entry (length = 560 bytes) per defect that is detected by LOAD (for example, conversion errors, unique index violations, violations of referential constraints).

Calculating the number of possible defects:

– For discard processing, if the discard limit is specified, the number of possible defects is equal to the discard limit.

  If the discard limit is the maximum, calculate the number of possible defects by using the following formula:

```
number of input records +
(number of unique indexes × number of extracted keys) +
(number of relationships × number of extracted foreign keys)
```

– For nondiscard processing, the data set is not required.

Allocating twice the space that is used by the input data sets is usually adequate for the sort work data sets. Two or three large SORTWK*nn* data sets are preferable to several small ones. For more information, see *DFSORT Application Programming: Guide.*

DB2 utilities uses DFSORT to perform sorts. Sort work data sets cannot span volumes. Smaller volumes require more sort work data sets to sort the same amount of data; therefore, large volume sizes can reduce the number of needed

sort work data sets. It is recommended that at least 1.2 times the amount of data to be sorted be provided in sort work data sets on disk. For more information about DFSORT, see *DFSORT Application Programming Guide*.

Instructions for specific tasks

The following tasks are described here:

Loading variable-length data

To load variable-length data, put a 2-byte binary length field before each field of variable-length data. The value in that field depends on the data type of the column into which you load the data. Use:

- The number of single-byte characters if the data type is VARCHAR
- The number of double-byte characters if the data type is VARGRAPHIC

For example, assume that you have a variable-length column that contains X'42C142C142C2', which might be interpreted as either six single-byte characters or three double-byte characters. With the two-byte length field, use:

- **X'0006'**X'42C142C142C2' to signify six single-byte characters in a VARCHAR column
- **X'0003'**X'42C142C142C2' to signify three double-byte characters in a VARGRAPHIC column

Ordering loaded records

The LOAD utility loads records into a table space in the order in which they appear in the input stream. It does not sort the input stream, and it does not insert records in sequence with existing records, even if a clustering index exists. To achieve clustering when loading an empty table or replacing data, sort the input stream. When adding data to a clustered table, consider reorganizing the table after running LOAD.

Because rows with duplicate key values for unique indexes fail to be loaded, any records that are dependent on such rows either:

- Fail to be loaded because they would cause referential integrity violations (if you specify ENFORCE CONSTRAINTS)
- Are loaded without regard to referential integrity violations (if you specify ENFORCE NO)

As a result, violations of referential integrity might occur. Such violations can be detected by LOAD (without the ENFORCE(NO) option) or by CHECK DATA.

Replacing data with LOAD

You can use LOAD REPLACE to replace data in a table space that has one or more tables. You can replace all the data in a table space (by using the REPLACE option), or you can load new records into a table space without destroying the rows already there (by using the RESUME option).

When you run a LOAD job with the REPLACE option but without the REUSE option and the data set that contains the data is not user-managed, DB2 deletes this data set before the LOAD and redefines a new data set with a control interval that matches the page size.

If an object is in REORG-pending status, you can perform a LOAD REPLACE of the entire table space (which resets REORG-pending status). You can also perform a LOAD PART REPLACE or RESUME of any partitions that are not in REORG-pending status. In this situation, no other LOAD operations are allowed. If an object is in advisory-REORG pending status (AREO*), you can perform a LOAD REPLACE of the entire table space (which resets advisory REORG-pending status).

If an object is in REBUILD-pending status, you can perform a LOAD REPLACE of the entire table space (which resets REBUILD-pending status). You can also perform a LOAD PART REPLACE or RESUME of any partitions. If these partitions are in REBUILD-pending status, a LOAD PART REPLACE or RESUME resets that status. If an object is in advisory REBUILD-pending status, you can perform a LOAD REPLACE of the entire table space (which resets advisory REBUILD-pending status). If a user-defined table space is in refresh-pending (REFP) status, you can replace the data by using LOAD REPLACE.

See Appendix C, “Advisory or restrictive states,” on page 853 for more information.

Using LOAD REPLACE with LOG YES: The LOAD REPLACE or PART REPLACE with LOG YES option logs only the reset and not each deleted row. If you need to see what rows are being deleted, use the SQL DELETE statement.

Replacing one table in a single-table table space: The control statement in Figure 33 on page 241 specifies that LOAD is to replace one table in a single-table table space:

```
LOAD DATA
  REPLACE
  INTO TABLE DSN8810.DEPT
  ( DEPTNO    POSITION (1)     CHAR(3),
    DEPTNAME  POSITION (5)     VARCHAR,
    MGRNO     POSITION (37)    CHAR(6),
    ADMRDEPT  POSITION (44)    CHAR(3),
    LOCATION  POSITION (48)    CHAR(16) )
  ENFORCE NO
```

Figure 33. Example of using LOAD to replace one table in a single-table table space

Replacing one table in a multiple-table table space: When using LOAD REPLACE on a multiple-table table space, you must be careful because LOAD works on an entire table space at a time. Thus, to replace all rows in a multiple-table table space, you must work with one table at a time, by using the RESUME YES option on all but the first table. For example, if you have two tables in a table space, you need to do the following steps:

1. Use LOAD REPLACE on the first table as shown in the control statement in Figure 34. This option removes data from the table space and replaces just the data for the first table.

```
LOAD DATA CONTINUEIF(72:72)='X'
  REPLACE
  INTO DSN8810.TOPTVAL
  ( MAJSYS    POSITION (2)    CHAR(1),
    ACTION    POSITION (4)    CHAR(1),
    OBJECT    POSITION (6)    CHAR(2),
    SRCHCRIT  POSITION (9)    CHAR(2),
    SCRTYPE   POSITION (12)   CHAR(1),
    HEADTXT   POSITION (80)   CHAR(50),
    SELTXT    POSITION (159)  CHAR(50),
    INFOTXT   POSITION (238)  CHAR(71),
    HELPTXT   POSITION (317)  CHAR(71),
    PFKTXT    POSITION (396)  CHAR(71),
    DSPINDEX  POSITION (475)  CHAR(2) )
```

Figure 34. Example of using LOAD REPLACE on the first table in a multiple-table table space

2. Use LOAD with RESUME YES on the second table as shown in the control statement in Figure 35. This option adds the records for the second table without destroying the data in the first table.

```
LOAD DATA CONTINUEIF(72:72)='X'
  RESUME YES
  INTO DSN8810.TDSPTXT
  ( DSPINDEX  POSITION (2)     CHAR(2),
    LINENO    POSITION (6)     CHAR(2),
    DSPLINE   POSITION (80)    CHAR(79) )
```

Figure 35. Example of using LOAD with RESUME YES on the second table in a multiple-table table space

If you need to replace just one table in a multiple-table table space, you need to delete all the rows in the table, and then use LOAD with RESUME YES. For example, assume that you want to replace all the data in DSN8810.TDSPTXT without changing any data in DSN8810.TOPTVAL. To do this, follow these steps:

1. Delete all the rows from DSN8810.TDSPTXT by using the following SQL DELETE statement:

```
EXEC SQL
   DELETE FROM DSN8810.TDSPTXT
ENDEXEC
```

Hint: The mass delete works most quickly on a segmented table space.

2. Use the LOAD job that is shown in Figure 36 to replace the rows in that table.

```
LOAD DATA CONTINUEIF(72:72)='X'
  RESUME YES
  INTO DSN8810.TDSPTXT
  ( DSPINDEX  POSITION (2)     CHAR(2),
    LINENO    POSITION (6)     CHAR(2),
    DSPLINE   POSITION (80)    CHAR(79) )
```

Figure 36. Example of using LOAD with RESUME YES to replace one table in a multiple-table table space

Using LOAD for tables with identity columns or ROWID columns

When you run the UNLOAD utility or the REORG utility with the UNLOAD EXTERNAL or DISCARD options, DB2 generates a LOAD statement that you can use to load the unloaded data into any table that has a compatible format. If the source table has a ROWID column that is defined with GENERATED ALWAYS, the generated LOAD statement contains a dummy field named DSN_ROWID for the ROWID column. If the source table has an identity column that is defined with GENERATED ALWAYS, the generated LOAD statement contains a dummy field named DSN_IDENTITY for the identity column. The keyword IGNOREFIELDS in the LOAD statement causes DB2 to skip the DSN_ROWID or DSN_IDENTITY field when it loads the data into a table. Using the combination of IGNOREFIELDS and the dummy fields, you can load the unloaded data into a compatible table that has GENERATED ALWAYS columns.

If you want to include the data from the identity column or ROWID column when you load the unloaded data into a table, the identity column or ROWID column in the target table must be defined with GENERATED BY DEFAULT. To use the generated LOAD statement, remove the IGNOREFIELDS keyword and change the dummy field names to the corresponding column names in the target table.

To load the unloaded data into a compatible table that has identity columns that are defined as GENERATED ALWAYS, use one of the following techniques:

- Using the combination of IGNOREFIELDS and the dummy DSN_IDENTITY field, load will generate the identity column data.
- To load the unloaded identity column data, add the IDENTITYOVERRIDE keyword to the LOAD control statement. Change the dummy field name, DSN_IDENTITY, to the corresponding identity column name in the target table.
- To load the unloaded data into a compatible table that has identity columns or ROWID columns that are defined as GENERATED BY DEFAULT, remove the IGNOREFIELDS keyword and change the dummy field names to the corresponding column names in the target table.
- To load the unloaded data into a compatible table that has ROWID columns that are defined as GENERATED ALWAYS, using the combintion of IGNOREFIELDS and the dummy DSN_ROWID field, load will generate the ROWID column data.

Adding more data to a table or partition

You might want to add data to a table, rather than replace it. The RESUME keyword specifies whether data is to be loaded into an empty or a non-empty table space. RESUME NO loads records into an empty table space. RESUME YES loads records into a non-empty table space.

If RESUME NO is specified and the target table is not empty, no data is loaded.

If RESUME YES is specified and the target table is empty, data is loaded.

LOAD always adds rows to the end of the existing rows, but index entries are placed in key sequence.

Deleting all the data in a table space

Specifying LOAD REPLACE without loading any records is an efficient way of clearing a table space. To achieve this, specify the input data set in the JCL as DD DUMMY. LOAD REPLACE is efficient for the following reasons:

1. LOAD REPLACE LOG NO does not log any rows.
2. LOAD REPLACE redefines the table space.
3. LOAD REPLACE retains all views and privileges that are associated with a table space or table.
4. LOG YES can be used to make the LOAD REPLACE recoverable.

LOAD REPLACE replaces ALL TABLES in the table space.

Loading partitions

If you use the PART clause of the INTO TABLE option, only the specified partitions of a partitioned table are loaded. If you omit PART, the entire table is loaded.

You can specify the REPLACE and RESUME options separately by partition. The control statement in Figure 37 specifies that DB2 is to load data into the first and second partitions of the employee table. Records with '0' in column 1 replace the contents of partition 1; records with '1' in column 1 are added to partition 2; all other records are ignored. (The example control statement, which is simplified to illustrate the point, does not list field specifications for all columns of the table.)

```
LOAD DATA CONTINUEIF(72:72)='X'
  INTO TABLE DSN8810.EMP PART 1 REPLACE WHEN (1) = '0'
  ( EMPNO     POSITION (1:6)  CHAR(6),
    FIRSTNME  POSITION (7:18) CHAR(12),

   ⋮
                                       )
  INTO TABLE DSN8810.EMP PART 2 RESUME YES WHEN (1) = '1'
  ( EMPNO     POSITION (1:6)  CHAR(6),
    FIRSTNME  POSITION (7:18) CHAR(12),

   ⋮
                                       )
```

Figure 37. Example LOAD control statement for loading partitions

If you are not loading columns in the same order as in the CREATE TABLE statement, you must code field specifications for each INTO TABLE statement.

The following example assumes that you have your data in separate input data sets. That data is already sorted by partition, so you do not need to use the WHEN clause of INTO TABLE. Placing the RESUME YES option before the PART option inhibits concurrent partition processing while the utility is running.

```
LOAD DATA INDDN EMPLDS1 CONTINUEIF(72:72)='X'
  RESUME YES
  INTO TABLE DSN8810.EMP REPLACE PART 1

LOAD DATA INDDN EMPLDS2 CONTINUEIF(72:72)='X'
  RESUME YES
  INTO TABLE DSN8810.EMP REPLACE PART 2
```

The following example allows partitioning independence when more than one partition is being loaded concurrently.

```
LOAD DATA INDDN SYSREC LOG NO
  INTO TABLE DSN8810.EMP PART 2 REPLACE
```

When index-based partitioning is used, LOAD INTO PART *integer* is not allowed if an identity column is part of the partitioning index. When table-based partitioning is used, LOAD INTO PART *integer* is not allowed if an identity column is used in a partitioning-clause of the CREATE TABLE or ALTER TABLE statement. If IDENTITYOVERRIDE is used, these operations are allowed.

To invoke partition parallelism, specify a PART clause with INDDN or INCURSOR and optionally DISCARDDN keywords for each partition in your utility control statement. This partition parallelism reduces the elapsed time that is required for loading large amounts of data into partitioned table spaces. Loading partition parallelism requires a separate input data set for each partition.

Consequences of DEFINE NO: If a partitioned table space is created with DEFINE NO, all partitions are also implicitly defined with DEFINE NO. The first data row that is inserted by the LOAD utility defines all data sets in the partitioned table space. If this process takes a long time, expect timeouts on the DBD.

Coding your LOAD job with SHRLEVEL CHANGE and using partition parallelism is equivalent to concurrent, independent insert jobs. For example, in a large partitioned table space that is created with DEFINE NO, the LOAD utility starts three tasks. The first task tries to insert the first row, which causes an update to the DBD. The other two tasks time out while they wait to access the DBD. The first task holds the lock on the DBD while the data sets are defined for the table space.

Loading delimited files

You can load a delimited file by using the FORMAT DELIMITED option. A delimited file contains cell values that are separated by delimiters. *Delimiters* are predefined characters that separate data. The column delimiter separates one column value from the next. Character string delimiters identify the beginning and end of a single cell value and are required only if the cell value contains the column delimiter. For more information about delimited files see Appendix F, “Delimited file format,” on page 899.

Recommendation: If a delimited file is to be transferred to or from a platform other than z/OS or between DB2 UDB for z/OS systems that use different EBCDIC or ASCII CCSIDs, use Unicode as the encoding scheme for the delimited file. Using Unicode avoids possible CCSID translation problems.

You are responsible for ensuring that the data in the file does not include the chosen delimiters. If the delimiters are part of the file's data, unexpected errors can occur.

Restrictions: The following restrictions apply to the use of delimiters:

- You cannot specify the same character for more than one type of delimiter (COLDEL, CHARDEL, and DECPT).
- You can specify a character constant for a delimiter if the utility control statement is coded in the same encoding scheme as the input file. For example, the utility control statement is coded in Unicode and the input data is also coded in Unicode.
- You should use the hex representation for non-default delimiters if the utility control statement is coded in a different encoding scheme than the input file. For example, the utility control statement is coded in Unicode and the input file is coded in EBCDIC. In this case, if you do not use the hex representation for the non-default delimiters, the results can be unpredictable.
- You do not need to specify the POSITION keyword when you specify the DELIMITED option. The utility ignores the POSITION keyword when you also specify DELIMITED. The utility overrides field data type specifications according to the specifications of the delimited format. (For example, length values for CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, CLOB, DBCLOB, and BLOB data are the delimited lengths of each field in the input data set, and the utility expects all numeric types in external format.)
- You cannot specify a binary 0 (zero) for any delimiter.
- You cannot specify the default decimal point as a string character delimiter (CHARDEL) or a column string delimiter (COLDEL).
- You cannot specify shift-in and shift-out characters for EBCDIC MBCS data.
- You cannot specify the pipe character (|) for DBCS data.

Table 32 lists the default hex values for the delimiter characters based on encoding scheme.

Table 32. Default delimiter values for different encoding schemes

Character	EBCDIC SBCS	EBCDIC DBCS/MBCS	ASCII/Unicode SBCS	ASCII/Unicode MBCS
Character string delimiter	X'7F'	X'7F'	X'22'	X'22'
Decimal point character	X'4B'	X'4B'	X'2E'	X'2E'
Column delimiter	X'6B'	X'6B'	X'2C'	X'2C'

Note: In most EBCDIC code pages, the hex values that are specified in Table 32 are a double quotation mark(") for the character string delimiter, a period(.) for the decimal point character, and a comma(,) for the column delimiter.

Table 33 lists the maximum allowable hex values for any delimiter character based on the encoding scheme.

Table 33. Maximum delimiter values for different encoding schemes

Encoding scheme	Maximum allowable value
EBCDIC SBCS	None

Table 33. Maximum delimiter values for different encoding schemes (continued)

Encoding scheme	Maximum allowable value
EBCDIC DBCS/MBCS	X'3F'
ASCII/Unicode SBCS	None
ASCII/Unicode MBCS	X'7F'

Table 34 identifies the acceptable data type forms for the delimited file format that the LOAD and UNLOAD utilities use.

Table 34. Acceptable data type forms for delimited files.

Data type	Acceptable form for loading a delimited file	Form that is created by unloading a delimited file
CHAR, VARCHAR	A delimited or non-delimited character string	Character data that is enclosed by character delimiters. For VARCHAR, length bytes do not precede the data in the string.
GRAPHIC (any type)	A delimited or non-delimited character stream	Data that is unloaded as a delimited character string. For VARGRAPHIC, length bytes do not precede the data in the string.
INTEGER (any type) 1	A stream of characters that represents a number in EXTERNAL format	Numeric data in external format.
DECIMAL (any type) 2	A character string that represents a number in EXTERNAL format	A string of characters that represents a number.
FLOAT 3	A representation of a number in the range -7.2E + 75 to 7.2E + 75 in EXTERNAL format	A string of characters that represents a number in floating-point notation.
BLOB, CLOB	A delimited or non-delimited character string	Character data that is enclosed by character delimiters. Length bytes do not precede the data in the string.
DBCLOB	A delimited or non-delimited character string	Character data that is enclosed by character delimiters. Length bytes do not precede the data in the string.
DATE	A delimited or non-delimited character string that contains a date value in EXTERNAL format	Character string representation of a date.
TIME	A delimited or non-delimited character string that contains a time value in EXTERNAL format	Character string representation of a time.

Table 34. Acceptable data type forms for delimited files. (continued)

Data type	Acceptable form for loading a delimited file	Form that is created by unloading a delimited file
TIMESTAMP	A delimited or non-delimited character string that contains a timestamp value in EXTERNAL format	Character string representation of a timestamp.

Note:

1. Field specifications of INTEGER or SMALLINT are treated as INTEGER EXTERNAL.
2. Field specifications of DECIMAL, DECIMAL PACKED, or DECIMAL ZONED are treated as DECIMAL EXTERNAL.
3. Field specifications of FLOAT, REAL, or DOUBLE are treated as FLOAT EXTERNAL.

Loading data with referential constraints

LOAD does not load a table with an incomplete definition; if the table has a primary key, the unique index on that key must exist. If any table that is to be loaded has an incomplete definition, the LOAD job terminates.

LOAD requires access to the primary indexes on the parent tables of any loaded tables. For simple, segmented, and partitioned table spaces, it drains all writers from the parent table's primary indexes. Other users cannot make changes to the parent tables that result in an update to their own primary indexes. Concurrent inserts and deletes on the parent tables are blocked, but updates are allowed for columns that are not defined as part of the primary index.

By default, LOAD enforces referential constraints, except informational referential constraints, which LOAD ignores. By enforcing referential constraints, LOAD provides you with several possibilities for error:

- Records that are to be loaded might have duplicate values of a primary key.
- Records that are to be loaded might have invalid foreign-key values, which are not values of the primary key of the corresponding parent table.
- The loaded table might lack primary key values that are values of foreign keys in dependent tables.

The next few paragraphs describe how DB2 signals each of those errors and the means it provides for correcting them.

Duplicate values of a primary key: A primary index must be a unique index and must exist if the table definition is complete. Therefore, when you load a parent table, you build at least its primary index. You need an error data set, and probably also a map data set and a discard data set.

Invalid foreign key values: A dependent table has the constraint that the values of its foreign keys must be values of the primary keys of corresponding parent tables. By default, LOAD enforces that constraint in much the same way as it enforces the uniqueness of key values in a unique index. First, it loads all records to the table. Subsequently, LOAD checks the validity of the records with respect to the constraints, identifies any invalid record by an error message, and deletes the record from the table. You can choose to copy this record to a discard data set. Again you need at least an error data set, and probably also a map data set and a discard data set.

If a record fails to load because it violates a referential constraint, any of its dependent records in the same job also fail. For example, suppose that the sample project table and project activity tables belong to the same table space, that you load them both in the same job, and that some input record for the project table has an invalid department number. Then, that record fails to be loaded and does not appear in the loaded table; the summary report identifies it as causing a **primary** error.

However the project table has a primary key, the project number. In this case, the record that is rejected by LOAD defines a project number, and any row in the project activity table that refers to the rejected number is also rejected. The summary report identifies those as causing **secondary** errors. If you use a discard data set, records for both types of errors are copied to it.

Missing primary key values: The deletion of invalid records does not cascade to other dependent tables that are already in place. Suppose now that the project and project activity tables exist in separate table spaces, and that they are both currently populated and possess referential integrity. In addition, suppose that the data in the project table is now to be replaced (using LOAD REPLACE) and that the replacement data for some department was inadvertently not supplied in the input data. Rows that reference that department number might already exist in the project activity table. LOAD, therefore, automatically places the table space that contains the project activity table (and all table spaces that contain dependent tables of any table that is being replaced) into CHECK-pending status.

The CHECK-pending status indicates that the referential integrity of the table space is in doubt; it might contain rows that violate a referential constraint. DB2 places severe restrictions on the use of a table space in CHECK-pending status; typically, you run the CHECK DATA utility to reset this status. For more information, see “Resetting the CHECK-pending status” on page 270.

Consequences of ENFORCE NO: If you use the ENFORCE NO option, you tell LOAD not to enforce referential constraints. Sometimes you have good reasons for doing that, but the result is that the loaded table space might violate the constraints. Hence, LOAD places the loaded table space in CHECK-pending status. If you use REPLACE, all table spaces that contain any dependent tables of the tables that were loaded are also placed in CHECK-pending status. You must reset the status of each table before you can use any of the table spaces.

Correcting referential constraint violations

The referential integrity checking in LOAD can delete only incorrect dependent rows, which were input to LOAD. Deletion is not always the best strategy for correcting referential integrity violations.

For example, the violations might occur because parent rows do not exist. In this case, correcting the parent tables is better than deleting the dependent rows. In this case, ENFORCE NO is more appropriate than ENFORCE CONSTRAINTS. After you correct the parent table, you can use CHECK DATA to reset the CHECK-pending status.

LOAD ENFORCE CONSTRAINTS is not equivalent to CHECK DATA. LOAD ENFORCE CONSTRAINTS deletes any rows that cause referential constraint violations. CHECK DATA detects violations and optionally deletes such rows. CHECK DATA checks a complete referential structure, although LOAD checks only the rows that are being loaded.

When loading referential structures with ENFORCE CONSTRAINTS, you should load tables before dependent tables.

Compressing data

You can use LOAD with the REPLACE, RESUME NO, or RESUME YES options to build a compression dictionary. The RESUME NO option requires the table space to be empty and RESUME YES will only build a dictionary if the table space is empty.

LOAD RESUME YES or NO will build compression dictionaries for empty table spaces, except for linear/simple table spaces. LOAD REPLACE must be used on linear/simple table spaces in order to build new compression dictionaries. If your table space, or a partition in a partitioned table space, is defined with COMPRESS YES, the dictionary is created while records are loaded. After the dictionary is completely built, the rest of the data is compressed as it is loaded.

The data is not compressed until the dictionary is built. You must use LOAD REPLACE or RESUME NO to build the dictionary, except for linear/simple table spaces where LOAD REPLACE must be used to build new compression dictionaries. To save processing costs, an initial LOAD does not go back to compress the records that were used to build the dictionary.

The number of records that are required to build a dictionary is dependent on the frequency of patterns in the data. For large data sets, the number of rows that are required to build the dictionary is a small percentage of the total number of rows that are to be compressed. For the best compression results, build a new dictionary whenever you load the data.

However, in some circumstances, you might want to compress data by using an existing dictionary. If you are satisfied with the compression that you are getting with an existing dictionary, you can keep that dictionary by using the KEEPDICTIONARY option of LOAD or REORG. For both LOAD and REORG, this method also saves you the processing overhead of building the dictionary. LOAD RESUME on a linear/simple table space will always keep the existing dictionary if one exists. In order to build new dictionaries for a linear/simple table space, LOAD REPLACE or REORG is required.

Consider using KEEPDICTIONARY if the last dictionary was built by REORG; the REORG utility's sampling method can yield more representative dictionaries than LOAD and can thus mean better compression. REORG with KEEPDICTIONARY is efficient because the data is not decompressed in the process.

However, REORG with KEEPDICTIONARY does not generate a compression report. You need to use RUNSTATS to update the catalog statistics and then query the catalog columns yourself. See Chapter 25, "REORG TABLESPACE," on page 417 for more information about using REORG to compress data, and see Chapter 29, "RUNSTATS," on page 551 for information about using RUNSTATS to update catalog information about compression.

Use KEEPDICTIONARY if you want to try to compress all the records during LOAD, and if you know that the data has not changed much in content since the last dictionary was built. An example of LOAD with the KEEPDICTIONARY option is shown in Figure 38 on page 250.

```
LOAD DATA
  REPLACE KEEPDICTIONARY
  INTO TABLE DSN8810.DEPT
  ( DEPTNO    POSITION (1)     CHAR(3),
    DEPTNAME  POSITION (5)     VARCHAR,
    MGRNO     POSITION (37)    CHAR(6),
    ADMRDEPT  POSITION (44)    CHAR(3),
    LOCATION  POSITION (48)    CHAR(16) )
  ENFORCE NO
```

Figure 38. Example of LOAD with the KEEPDICTIONARY option

You can also specify KEEPDICTIONARY for specific partitions of a partitioned table space. In this case, each partition has its own dictionary.

Loading data from DL/I

To convert data in IMS DL/I databases from a hierarchic structure to a relational structure so that it can be loaded into DB2 tables, you can use the DataRefresher and IMS DataPropagator (IMS DPROP) licensed programs. You can use DataRefresher to create source-to-target mappings and to create DB2 databases. After your databases are created and the mappings are set, you can use IMS DPROP to propagate any changes.

IMS DPROP runs as a z/OS application and can extract data from VSAM and physical sequential access method (SAM) files, as well from DL/I databases. Using IMS DPROP, you do not need to extract all the data in a database or data set. You use a statement such as an SQL subselect to indicate which fields to extract and which conditions, if any, the source records or segments must meet.

With JCL models that you edit, you can have IMS DPROP produce the statements for a DB2 LOAD utility job. If you have more than one DB2 subsystem, you can name the one that is to receive the output. IMS DPROP can generate LOAD control statements in the job to relate fields in the extracted data to target columns in DB2 tables.

You have the following choices for how IMS DPROP writes the extracted data:

- 80-byte records, which are included in the generated job stream
- A separate physical sequential data set (which can be dynamically allocated by IMS DPROP), with a logical record length that is long enough to accommodate any row of the extracted data

In the first case, the LOAD control statements that are generated by IMS DPROP include the CONTINUEIF option to describe the extracted data to DB2 LOAD.

In the second case, you can have IMS DPROP name the data set that contains the extracted data in the SYSREC DD statement in the LOAD job. (In that case, IMS DPROP makes no provision for transmitting the extracted data across a network.)

Normally, you do not need to edit the job statements that are produced by IMS DPROP. However, in some cases you might need to edit; for example, if you want to load character data into a DB2 column with INTEGER data type, you need to edit the job statements. (DB2 LOAD does not consider CHAR and INTEGER data to be compatible.)

IMS DPROP is a versatile tool that contains more control, formatting, and output options than are described here. For more information about this tool, see *IMS DataPropagator: An Introduction.*

Loading data by using the cross-loader function

The LOAD utility can directly load the output of a dynamic SQL SELECT statement into a table. The dynamic SQL statement can be executed on data at a local server or at any DRDA-compliant remote server. This functionality is called the DB2 UDB family cross-loader function. This function enables you to use a single LOAD job to transfer data from one location to another location or from one table to another table at the same location. Your input for this cross-loader function can come from other sources besides DB2 UDB for z/OS; you can use IBM Information Integrator Federation feature for access to data from sources as diverse as Oracle and Sybase, as well as the entire DB2 UDB family of database servers. For information about steps that need to be completed prior to using the cross loader function, see "Loading data by using a cursor" on page 235.

To use the cross-loader function, you first need to declare a cursor by using the EXEC SQL utility. Within the cursor definition, specify a SELECT statement that identifies the result table that you want to use as the input data for the LOAD job. The column names in the SELECT statement must be identical to the column names in the table that is being loaded. You can use the AS clause in the SELECT list to change the columns names that are returned by the SELECT statement so that they match the column names in the target table. The columns in the SELECT list do not need to be in the same order as the columns in the target table. Also, the SELECT statement needs to refer to any remote tables by their three-part name.

After you declare the cursor, specify the cursor name with the INCURSOR option in the LOAD statement. You cannot load the input data into the same table on which you defined the cursor. You can, however, use the same cursor to load multiple tables.

When you submit the LOAD job, DB2 parses the SELECT statement in the cursor definition and checks for errors. If the statement is invalid, the LOAD utility issues an error message and identifies the condition that prevented the execution. If the statement syntax is valid but an error occurs during execution, the LOAD utility also issues an error message. The utility terminates when it encounters an error.

If no errors occur, the utility loads the result table that is identified by the cursor into the specified target table according to the following rules:

- LOAD matches the columns in the input data to columns in the target table by name, not by sequence.
- If the number of columns in the cursor is less than the number of columns in the table that is being loaded, DB2 loads the missing columns with their default values. If the missing columns are defined as NOT NULL without defaults, the LOAD job fails.
- If you specify IGNOREFIELDS YES, LOAD skips any columns in the input data that do not exist in the target table.
- If the data types in the target table do not match the data types in the cursor, DB2 tries to convert the data as much as possible. If the conversion fails, the LOAD job fails. You might be able to avoid these conversion errors by using SQL conversion functions in the SELECT statement of the cursor declaration.
- If the encoding scheme of the input data is different than the encoding scheme of the target table, DB2 converts the encoding schemes automatically.
- The sum of the lengths of all of the columns cannot exceed 1 GB.

- If the SELECT statement in the cursor definition specifies a table with at least one LOB column and a ROWID that was created with the GENERATED ALWAYS clause, you cannot specify this ROWID column in the SELECT list of the cursor.

Also, although you do not need to specify casting functions for any distinct types in the input data or target table, you might need to add casting functions to any additional WHERE clauses in the SQL.

For examples of loading data from a cursor, see "Sample LOAD control statements" on page 274.

Using inline COPY with LOAD

You can create a full image copy data set (SHRLEVEL REFERENCE) during LOAD execution. The new copy is an inline copy. The advantage to using an inline copy is that the table space is not left in COPY-pending status regardless of which LOG option was specified for the utility. Thus, data availability is increased.

To create an inline copy, use the COPYDDN and RECOVERYDDN keywords. You can specify up to two primary and two secondary copies. Inline copies are produced during the RELOAD phase of LOAD processing.

The SYSCOPY record that is produced by an inline copy contains ICTYPE=F and SHRLEVEL=R. The STYPE column contains an R if the image copy was produced by LOAD REPLACE LOG(YES). It contains an S if the image copy was produced by LOAD REPLACE LOG(NO). The data set that is produced by the inline copy is logically equivalent to a full image copy with SHRLEVEL REFERENCE, but the data within the data set differs in the following ways:

- Data pages might be out of sequence and some might be repeated. If pages are repeated, the last one is always the correct copy.
- Space map pages are out of sequence and might be repeated.
- If the compression dictionary is rebuilt with LOAD, the set of dictionary pages occurs twice in the data set, with the second set being the correct one.

The total number of duplicate pages is small, with a negligible effect on the required space for the data set.

You must specify LOAD REPLACE. If you specify RESUME YES or RESUME NO but not REPLACE, an error message is issued and LOAD terminates.

Improving performance

To improve LOAD utility performance, you can take the following actions:

- Use one LOAD DATA statement when loading multiple tables in the same table space. Follow the LOAD statement with multiple INTO TABLE WHEN clauses.
- Run LOAD concurrently against separate partitions of a partitioned table space.

  Alternatively, specify the INDDN and DISCARDDN keywords in your utility JCL to invoke partition parallelism. This specification reduces the elapsed time required for loading large amounts of data into partitioned table spaces.

  Recommendation: Use load partition parallelism to load all partitions in a single job when one or more nonpartitioned secondary indexes exists. If the only indexes are the partitioned indexes, using multiple concurrent jobs against separate partitions is better.
- Preprocess the input data. For more information about preprocessing input data, see "Before running LOAD" on page 234.

- Load numeric data in its internal representation.
- Avoid data conversion, such as from integer to decimal or from decimal to floating-point.
- When you specify LOAD REPLACE, specify LOG NO with COPYDDN or RECOVERYDDN to create an inline copy.
- Sort the data in cluster order to avoid needing to reorganize it after loading.
- If you are using 3990 caching, and you have the secondary indexes on RAMAC , consider specifying YES on the UTILITY CACHE OPTION field of installation panel DSNTIPE. This allows DB2 to use sequential prestaging when reading data from RAMAC for the following utilities:
  – LOAD PART *integer* RESUME
  – REORG TABLESPACE PART

  For these utilities, prefetch reads remain in the cache longer, thus possibly improving performance of subsequent writes.

The optimum order for presenting data to LOAD is as follows:

- If you are loading a single table that has, at most, one foreign key or one index key, sort the data in key sequence. (An index over a foreign key is allowed.) If the key is an index key, sort the data in either ascending or descending order, depending on how the index was defined. If the key is a foreign key, sort the data in ascending order. Null key values are treated as “high” values.
- If you are loading more than one table, choose one of the following methods:
  – Load each table separately. Using this method, you can follow the rules listed in the preceding bullet for loading single tables.
  – Use the WHEN clause under each INTO TABLE option on your LOAD statement to group your input data by table.

  Within each table, sort the data in key sequence.

Improving performance for parallel processing

Taking advantage of any new parallelism feature without allocating additional resources or tuning your system can lead to significant performance degradation. To benefit from parallel operations when using LOAD SHRLEVEL CHANGE or parallel inserts, especially when secondary indexes are used, you can take the following actions:

- Use a larger buffer pool to improve the buffer-pool hit ratio.
- Define a higher deferred-write threshold to reduce the number of pages that are written to disk, which reduces the I/O time and contention.
- Define a larger checkpoint interval to reduce the number of pages that are written to disk, which reduces the I/O time and contention.
- Use ESS Parallel Access Volume (PAV) to support multiple concurrent I/Os to the same volume that contains secondary index data sets.
- Use secondary index pieces to support multiple concurrent secondary index I/Os.

Improved performance with SORTKEYS

The SORTKEYS keyword improves performance of the index key sort. The SORTKEYS keyword is the default if one of the following conditions is true:

- SHRLEVEL is not NONE.
- SHRLEVEL is NONE, and the target table has one or more indexes.

Advantages of the SORTKEYS option: With SORTKEYS, index keys are passed in memory rather than written to work files. Avoiding this I/O to the work files improves LOAD performance.

You also reduce disk space requirements for the SYSUT1 and SORTOUT data sets, especially if you provide an estimate of the number of keys to sort.

The SORTKEYS option reduces the elapsed time from the start of the RELOAD phase to the end of the BUILD phase.

You can reduce the elapsed time of a LOAD job for a table space or partition with more than one defined index by specifying the parameters to invoke a parallel index build. For more information, see "Building indexes in parallel for LOAD" on page 258.

Estimating the number of keys: You can specify an estimate of the number of keys for the job to sort. If the estimate is specified as 0, LOAD writes the extracted keys to the work data set, which reduces the performance improvement of using SORTKEYS.

To estimate the number of keys to sort:

1. Count 1 for each index.
2. Count 1 for each foreign key where foreign key and index definitions are not identical.
3. For each foreign key where foreign key and index definitions are identical:
   a. Count 0 for the first relationship in which the foreign key participates.
   b. Count 1 for subsequent relationships in which the foreign key participates (if any).
4. Multiply the count by the number of rows to be loaded.

| If more than one table is being loaded, repeat the preceding steps for each table,
| and sum the results.

Improving performance with LOAD or REORG PREFORMAT

DB2 preformatting sometimes causes delay, which can affect the performance or
execution time consistency of high INSERT applications or LOAD jobs with
RESUME YES SHRLEVEL CHANGE. These LOAD jobs are also referred to as
online LOAD jobs.. When these delays occur and when you can predict the table size for a business processing cycle, consider the LOAD PREFORMAT or REORG PREFORMAT technique. This technique is of value only when DB2 preformatting causes a measurable delay with processing or causes inconsistent application
elapsed times for INSERT or online LOAD jobs.

Recommendation: Assess performance before and after using LOAD or REORG PREFORMAT to quantify its value in your environment.

Considerations for using PREFORMAT: PREFORMAT is a technique that is used to eliminate the need for DB2 to preformat new pages in a table space during execution time. This technique might eliminate execution time delays but adds setup time prior to the application's execution. LOAD or REORG PREFORMAT
primes a new table space and prepares it for INSERT or online LOAD processing. When the preformatted space is utilized and DB2 needs to extend the table space, normal data set extending and preformatting occurs.

Preformatting for online LOAD or INSERT processing can be desirable for high-insert tables that receive a predictable amount of data because all the required space can be pre-allocated prior to the application's execution. This benefit also applies to the case of a table that acts as a repository for work items that come into a system and that are subsequently used to feed a backend task that processes the work items.

Preformatting of a table space that contains a table that is used for query processing can cause table space scans to read additional empty pages, extending the elapsed time for these queries. LOAD or REORG PREFORMAT is not recommended for tables that have a high ratio of reads to inserts if the reads result in table space scans.

Preformatting boundaries: You can manage your own data sets or have DB2 manage the data sets. For user-managed data sets, DB2 does not delete and reallocate them during utility processing. The size of the data set does not shrink back to the original data set allocation size but either remains the same or increases in size if additional space or data is added. This characteristic has implications when LOAD or REORG PREFORMAT is used because of the preformatting that is done for all free pages between the high-used RBA (or page) to the high-allocated RBA. This preformatting includes secondary extents that have been allocated.

For DB2-managed data sets, DB2 deletes and reallocates them if you specify REPLACE on the LOAD or REORG job. This results in the data sets being re-sized to their original allocation size. They remain that size if the data that is being reloaded does not fill the primary allocation and forces a secondary allocation. This means the LOAD or REORG PREFORMAT option with DB2-managed data causes at least the full primary allocation amount of a data set to be preformatted after the reload of data into the table space.

For both user-managed and DB2-managed data sets, if the data set goes into secondary extents during utility processing, the high-allocated RBA becomes the end of the secondary extent, and that becomes the high value for preformatting.

Preformatting performance considerations: LOAD or REORG PREFORMAT can eliminate dynamic preformatting delays when inserting into a new table space. The cost of this execution time improvement is an increase in the LOAD or REORG time due to the additional required processing to preformat all pages between the data that is loaded or reorganized and the high-allocated RBA. The additional LOAD or REORG time that is required depends on the amount of disk space that is being preformatted.

Table space scans can also be elongated because empty preformatted pages are read. Use the LOAD or REORG PREFORMAT option for table spaces that start out empty and are filled through high insert activity before any query access is performed against the table space. Mixing inserts and nonindexed queries against a preformatted table space might have a negative impact on the query performance without providing a compensating improvement in the insert performance. You will see the best results where a high ratio of inserts to read operations exists.

Converting input data

The LOAD utility converts data between compatible data types.[2]

2. The source type is used for user-defined distinct types.

Tables 35, 36, and 37 identify the compatibility of data types for assignments and comparisons. Y indicates that the data types are compatible. N indicates that the data types are not compatible. D indicates the defaults that are used when you do not specify the input data type in a field specification of the INTO TABLE statement.

Table 35 shows the compatibility of numeric data types.

Table 35. Compatibility of converting numeric data types.

Input data types	Output data types			
	SMALLINT	INTEGER	DECIMAL	FLOAT
SMALLINT	D	Y	Y	Y
INTEGER	Y	D	Y	Y
DECIMAL	Y	Y	D	Y
FLOAT	Y	Y	Y	D

Table 36 shows the compatibility of character data types.

Table 36. Compatibility of converting character data types.

Input data types	Output data types							
	BLOB	CHAR	VAR-CHAR	CLOB	GRAPHIC	VAR-GRAPHIC	DBCLOB	ROWID
CHAR	Y	D	Y	Y	Y[1]	Y[1]	Y[1]	Y
CHAR MIXED	Y	D	Y	Y	Y[1]	Y[1]	Y[1]	N
VAR- CHAR	Y	Y	D	Y	Y[1]	Y[1]	Y[1]	Y
VAR- CHAR MIXED	Y	Y	D	Y	Y[1]	Y[1]	Y[1]	N
GRAPHIC	N	Y[1]	Y[1]	Y[1]	D	Y	Y	N
VAR-GRAPHIC	N	Y[1]	Y[1]	Y[1]	Y	D	Y	N
ROWID	N	N	N	N	N	N	N	D

Notes:

1. Conversion applies when either the input data or the target table is Unicode.

Table 37 shows the compatibility of time data types.

Table 37. Compatibility of converting time data types.

Input data types	Output data types		
	DATE	TIME	TIMESTAMP
DATE EXTERNAL	D	N	N
TIME EXTERNAL	N	D	N
TIMESTAMP EXTERNAL	Y	Y	D

Input fields with data types CHAR, CHAR MIXED, CLOB, DBCLOB, VARCHAR, VARCHAR MIXED, GRAPHIC, GRAPHIC EXTERNAL, and VARGRAPHIC are converted from the CCSIDs of the input file to the CCSIDs of the table space when they do not match. For example:

- You specify the ASCII or UNICODE option for the input data, and the table space is EBCDIC.
- You specify the EBCDIC or UNICODE option, and the table space is ASCII.
- You specify the ASCII or EBCDIC option, and the table space is Unicode.
- The CCSID option is specified, and the CCSIDs of the input data are not the same as the CCSIDs of the table space.

CLOB, BLOB, and DBCLOB input field types cannot be converted to any other field type.

Conversion errors cause LOAD:

- To abend, if no discard data set is provided or if the discard limit is exceeded.
- To map the input record for subsequent discarding and continue (if a discard data set is provided)

Truncation of the decimal part of numeric data is not considered a conversion error.

Specifying input fields

When specifying input fields, take one of these actions:

- Specify the length of VARCHAR, BLOB, CLOB, DBCLOB, and ROWID data in the input file.
- Explicitly define all input field specifications.
- Use DECIMAL EXTERNAL(*length*,*scale*) in full.
- Specify decimal points explicitly in the input file.

Specifying the TRUNCATE and STRIP options

You can load certain fields that are longer than the length of target column by truncating the data. DB2 truncates the data only when you explicitly specify the TRUNCATE option. You can specify TRUNCATE with the CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data type options. LOAD first applies any CCSID conversion, and then truncates the data. The TRUNCATE option of the LOAD utility truncates string data, and it has a different purpose than the SQL TRUNCATE scalar function.

You can also remove a specified character from the beginning, end, or both ends of the data by specifying the STRIP option. This option is 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. For example, if you specify both TRUNCATE and STRIP for a field that is to be loaded into a VARCHAR(5) column, LOAD alters the character strings as shown in Table 38. In this table, an underscore represents a character that is to be stripped.

Table 38. Results of specifying both TRUNCATE and STRIP for data that is to be loaded into a VARCHAR(5) column.

Specified STRIP option	Input string	String after strip operation	String that is loaded
STRIP BOTH	'_ABCDEFG_'	'ABCDEFG'	'ABCDE'

Table 38. Results of specifying both TRUNCATE and STRIP for data that is to be loaded into a VARCHAR(5) column. (continued)

Specified STRIP option	Input string	String after strip operation	String that is loaded
STRIP LEADING	'_ABC_'	'ABC_'	'ABC_'
STRIP TRAILING	'_ABC_DEF_'	'_ABC_DEF'	'_ABC_'

Building indexes while loading data

LOAD builds all the indexes that are defined for any table that is being loaded. At the same time, it checks for duplicate values of any unique index key. If LOAD finds any duplicate values, none of the corresponding rows are loaded. Error messages identify the input records that produce duplicates; optionally, the records are copied to a discard data set. At the end of the job, a summary report lists all errors that are found.

For unique indexes, any two null values are assumed to be equal, unless the index was created with the UNIQUE WHERE NOT NULL clause. In that case, if the key is a single column, it can contain any number of null values, although its other values must be unique.

Neither the loaded table nor its indexes contain any of the records that might have produced an error. Using the error messages, you can identify faulty input records, correct them, and load them again. If you use a discard data set, you can correct the records there and add them to the table with LOAD RESUME.

Building indexes in parallel for LOAD

Parallel index build reduces the elapsed time for a LOAD job by sorting the index keys and rebuilding multiple indexes in parallel, rather than sequentially. Optimally, a pair of subtasks process each index; one subtask sorts extracted keys while the other subtask builds the index. LOAD begins building each index as soon as the corresponding sort produces its first sorted record. For more information about improving index key sort performance, see "Improved performance with SORTKEYS" on page 253.

LOAD uses parallel index build if all of the following conditions are true:

- More than one index needs to be built.
- The LOAD utility statement specifies a non-zero estimate of the number of keys on the SORTKEYS option.

For a diagram of parallel index build processing, see Figure 78 on page 473.

You can either allow the utility to dynamically allocate the data sets that the SORT phase needs, or provide the necessary data sets yourself. Select one of the following methods to allocate sort work and message data sets:

Method 1: LOAD determines the optimal number of sort work and message data sets.

1. Specify the SORTDEVT keyword in the utility statement.
2. Allow dynamic allocation of sort work data sets by **not** supplying SORTWK*nn* DD statements in the LOAD utility JCL.
3. Allocate UTPRINT to SYSOUT.

Method 2: You control allocation of sort work data sets, while LOAD allocates message data sets.

1. Provide DD statements with DD names in the form SW*nn*WK*mm*.
2. Allocate UTPRINT to SYSOUT.

Method 3: You have the most control over rebuild processing; you must specify both sort work and message data sets.

1. Provide DD statements with DD names in the form SW*nn*WK*mm*.
2. Provide DD statements with DD names in the form UTPRIN*nn*.

Note: Using this method does not eliminate the requirement for a UTPRINT DD card.

Data sets used: If you select Method 2 or 3 in the preceding information, use the information provided here, along with "Determining the number of sort subtasks," "Allocation of sort subtasks" on page 260, and "Estimating the sort work file size" on page 260 to define the necessary data sets.

Each sort subtask must have its own group of sort work data sets and its own print message data set. Possible reasons to allocate data sets in the utility job JCL rather than using dynamic allocation are:

- To control the size and placement of the data sets
- To minimize device contention
- To optimally utilize free disk space
- To limit the number of utility subtasks that are used to build indexes

The DD names SW*nn*WK*mm* define the sort work data sets that are used during utility processing. *nn* identifies the subtask pair, and *mm* identifies one or more data sets that are to be used by that subtask pair. For example:

SW01WK01 The first sort work data set that is used by the subtask as it builds the first index.

SW01WK02 The second sort work data set that is used by the subtask as it builds the first index.

SW02WK01 The first sort work data set that is used by the subtask as it builds the second index.

SW02WK02 The second sort work data set that is used by the subtask as it builds the second index.

The DD names UTPRIN*nn* define the sort work message data sets that are used by the utility subtask pairs. *nn* identifies the subtask pair.

Determining the number of sort subtasks: The maximum number of utility subtask pairs that are started for parallel index build is equal to the number of indexes that are to be built.

LOAD determines the number of subtask pairs according to the following guidelines:

- The number of subtask pairs equals the number of sort work data set groups that are allocated.
- The number of subtask pairs equals the number of message data sets that are allocated.

- If you allocate both sort work and message data set groups, the number of subtask pairs equals the smallest number of data sets that are allocated.

Allocation of sort subtasks: LOAD attempts to assign one sort subtask pair for each index that is to be built. If LOAD cannot start enough subtasks to build one index per subtask pair, it allocates any excess indexes across the pairs (in the order that the indexes were created), so that one or more subtask pairs might build more than one index.

During parallel index build processing, LOAD assigns all foreign keys to the first utility subtask pair. Remaining indexes are then distributed among the remaining subtask pairs according to the creation date of the index. If a table space does not participate in any relationships, LOAD distributes all indexes among the subtask pairs according to the index creation date, assigning the first created index to the first subtask pair.

Refer to Table 39 for conceptual information about subtask pairing when the number of indexes (seven indexes) exceeds the available number of subtask pairs (five subtask pairs).

Table 39. LOAD subtask pairing for a relational table space

Subtask pair	Assigned index
SW01WK*mm*	Foreign keys, fifth created index
SW02WK*mm*	First created index, sixth created index
SW03WK*mm*	Second created index, seventh created index
SW04WK*mm*	Third created index
SW05WK*mm*	Fourth created index

Estimating the sort work file size: If you choose to provide the data sets, you need to know the size and number of keys in all of the indexes that are being processed by the subtask in order to calculate each sort work file size. After you determine which indexes are assigned to which subtask pairs, use one of the following formulas to calculate the required space:

- If the indexes being processed include a mixture of data-partitioned secondary indexes and nonpartitioned indexes, use the following formula:
  2 × (longest index key + 15) × (number of extracted keys)
- Otherwise, if only one type of index is being built, use the following formula:
  2 × (longest index key + 13) × (number of extracted keys)

longest index key — The length of the longest key that is to be processed by the subtask. For the first subtask pair for LOAD, compare the length of the longest key and the length of the longest foreign key, and use the larger value. For nonpadded indexes, longest index key means the maximum possible length of a key with all varying-length columns, padded to their maximum lengths, plus 2 bytes for each varying-length column.

number of extracted keys — The number of keys from all indexes that are to be sorted and that the subtask is to process.

Leaving free space

When loading into a nonsegmented table space, LOAD leaves one free page after reaching the FREEPAGE limit, regardless of whether the loaded records belong to the same or different tables.

When loading into a segmented table space, LOAD leaves free pages, and free space on each page, in accordance with the current values of the FREEPAGE and PCTFREE parameters. (You can set those values with the CREATE TABLESPACE, ALTER TABLESPACE, CREATE INDEX, or ALTER INDEX statements.) LOAD leaves one free page after reaching the FREEPAGE limit for each table in the table space.

Loading with RECOVER-pending, REBUILD-pending, or REORG-pending status

You cannot load records by specifying RESUME YES if any partition of a table space is in the RECOVER-pending status. In addition, you cannot load records if any index on the table that is being loaded is in the REBUILD-pending status. See "Resetting the REBUILD-pending status" on page 348 for information about resetting the REBUILD-pending status.

If you are replacing a partition, these preceding restrictions are relaxed; the partition that is being replaced can be in the RECOVER-pending status, and its corresponding index partition can be in the REBUILD-pending status. However, all secondary indexes must **not** be in the page set REBUILD-pending status. See Appendix C, "Advisory or restrictive states," on page 853 for more information about resetting a restrictive status.

The one RECOVER-pending restrictive status has the following description:

RECP RECOVER-pending status is set on a table space or partition. If a single logical partition is in RECP status, the partition is treated as RECP status for SQL access. A single logical partition in RECP status does not restrict utility access to other logical partitions that are not in RECP status. RECP status is reset by recovering only the single logical partition.

The four REBUILD-pending restrictive states have the following descriptions:

RBDP REBUILD-pending status is set on a physical or logical index partition. The individual physical or logical partition is inaccessible and must be rebuilt by using the REBUILD INDEX utility, or recovered by using the RECOVER utility.

PSRBD
Page set REBUILD-pending is set on nonpartitioned secondary indexes. Partitioned indexes, including data-partitioned secondary indexes, are never placed in a page set REBUILD-pending status. The entire index space is inaccessible until you rebuild it with the REBUILD utility, or recover it with the RECOVER utility.

RBDP*
REBUILD-pending star status is set only on logical partitions of nonpartitioning indexes. The entire index is inaccessible, but it is made available again when the affected partitions are rebuilt by using the REBUILD INDEX utility, or recovered by using the RECOVER utility.

The one REORG-pending restrictive status has the following description:

REORP
REORG-pending status indicates that a table space or partition needs to be reorganized.

See Table 171 on page 857 for information about resetting the RECOVER-pending status, Table 170 on page 857 for information about resetting the REBUILD-pending status, and "REORG-pending status" on page 858 for information about resetting the REORG-pending status.

Using exit procedures

Any field procedure that is associated with a column of a table that is being loaded is executed to encode the data before it is loaded. The field procedures for all columns are executed before any edit or validation procedure for the row.

Any field specification that describes the data is checked before a field procedure is executed. That is, the field specification must describe the data as it appears in the input record.

Loading ROWID columns

Columns that are defined as ROWID can be designated as input fields; refer to the LOAD field specification syntax diagram. LOAD PART is not allowed if the ROWID column is part of the partitioning key. In this situation, DB2 issues error message DSNU256I.

Columns that are defined as ROWID can be designated as GENERATED BY DEFAULT or GENERATED ALWAYS. With GENERATED ALWAYS, DB2 always generates a row ID.

ROWID generated by default: The LOAD utility can set from input data columns that are defined as ROWID GENERATED BY DEFAULT. The input field must be specified as a ROWID. No conversions are allowed. The input data for a ROWID column must be a unique, valid value for a row ID. If the value of the row is not unique, a duplicate key violation occurs. If such an error occurs, the load fails. In this case, you need to discard the duplicate value and re-run the LOAD job with a new unique value, or allow DB2 to generate the value of the row ID.

You can use the DEFAULTIF attribute with the ROWID keyword. If the condition is met, the column is loaded with a value that is generated by DB2. You cannot use the NULLIF attribute with the ROWID keyword because row ID columns cannot be null.

ROWID generated always: A ROWID column that is defined as GENERATED ALWAYS cannot be included in the field specification list because DB2 generates the row ID value for you.

Loading a LOB column

LOB columns are treated by the LOAD utility as varying-length data. The length value for a LOB column must be 4 bytes.

You can load LOB values in one of the following ways:

- **Load the LOB value directly from the input data set:** Use this method only when the sum of the lengths of all of the columns to be loaded, including the LOB column, does not exceed 32 KB. To load a LOB value directly from the input data set:
  1. In the input data set, include the LOB value preceded by a 4-byte binary field that contains the length of the LOB.

2. Specify CLOB, BLOB, or DBCLOB in the field specification portion of the LOAD statement. These options indicate that the field in the input data set is a LOB value. For example, to load a CLOB into the RESUME column, specify something like `RESUME POSITION(7) CLOB`. This specification indicates that position 7 of the input data set contains the length of the CLOB followed by the CLOB value that is to be loaded into the RESUME column.

- **Load the LOB value from a file that is listed in the input data set:** When you load a LOB value from a file, the LOB value can be greater than 32 KB. To load a LOB value from a file:
  1. In the input data set, specify the names of the files that contain the LOB values. Each file can be either a PDS, PDSE, or an HFS file.
  2. Specify either BLOBF, CLOBF, or DBCLOBF in the field specification portion of the LOAD statement. For example, to load a LOB into the RESUME column of a table, specify something like `RESUME POSITION(7) VARCHAR CLOBF`. This specification indicates that position 7 of the input data set contains the name of a file from which a varying-length CLOB is to be loaded into the RESUME column.
- **Load data from another table:** To transfer data from one location to another location or from one table to another table at the same location, use a cursor. This method of loading data is called the cross-loader function. For more information about how to use this function, see "Loading data by using the cross-loader function" on page 251.

  When you use the cross-loader function, the LOB value can be greater than 32 KB. For this method, DB2 uses a separate buffer for LOB data and therefore stores only 8 bytes per LOB column. The sum of the lengths of the non-LOB columns plus the sum of 8 bytes per LOB column cannot exceed 32 KB.

Using LOAD LOG on a LOB table space

A LOB table space that was defined with LOG YES or LOG NO affects logging during the load of a LOB column. Table 40 shows the logging output and LOB table space effect, if any.

Table 40. LOAD LOG and REORG LOG impact for a LOB table space

LOAD LOG/ REORG LOG keyword	LOB table space LOG attribute	What is logged	LOB table space status after utility completes
LOG YES	LOG YES	Control information and LOB data	No pending status
LOG YES	LOG NO	Control information	No pending status
LOG NO	LOG YES	Nothing	COPY-Pending[1]
LOG NO	LOG NO	Nothing	COPY-Pending[1]

Notes:

1. REORG LOG NO on a LOB table space sets COPY-pending status only if the LOB table space was changed by the REORG utility.

Collecting inline statistics while loading a table

If you do not specify LOAD RESUME YES, you can use the STATISTICS keyword to gather inline statistics. Using the STATISTICS keyword eliminates the need to run RUNSTATS after loading a table space. However, if you perform a LOAD PART operation, you should run RUNSTATS INDEX on the nonpartitioned secondary indexes to update the catalog data about these indexes.

Use either the STATISTICS option or the RUNSTATS utility to collect statistics so that the DB2 catalog statistics contain information about the newly loaded data. Recording these new statistics enables DB2 to select SQL paths with accurate information. Then rebind any application plans that depend on the loaded tables to update the path selection of any embedded SQL statements.

Collecting inline statistics for discarded rows: If you specify the DISCARDDN and STATISTICS options and a row is found with check constraint errors or conversion errors, the row is not loaded into the table and DB2 does not collect inline statistics on it. However, the LOAD utility collects inline statistics prior to discarding rows that have unique index violations or referential integrity violations. In these cases, if the number of discarded rows is large enough to make the statistics significantly inaccurate, run the RUNSTATS utility separately on the table to gather the most accurate statistics.

Collecting inline statistics for data partitioned secondary indexes: To collect inline statistics on data partitioned secondary indexes, you must allocate sort work data sets. For information about these data sets, including how to estimate the space, see "Data sets that LOAD uses" on page 235.

Inline COPY for a base table space

If you take an inline image copy of a table that has LOB columns, DB2 makes a copy of the base table space, but does not copy the LOB table spaces.

Terminating or restarting LOAD

This section contains information about how to terminate and restart LOAD.

Terminating LOAD

If you terminate LOAD by using the TERM UTILITY command during the reload phase, the records are not erased. The table space remains in RECOVER-pending status, and indexes remain in the REBUILD-pending status.

If you terminate LOAD by using the TERM UTILITY command during the sort or build phases, the indexes that are not yet built remain in the REBUILD-pending status.

If you terminate a LOAD SHRLEVEL CHANGE, uncommitted records are rolled back, but committed records remain in the table. The table space is not in RECOVER-pending status, and the indexes are not in REBUILD-pending status.

If the LOAD job terminates during the RELOAD, SORT, BUILD, or SORTBLD phases, both RESTART and RESTART(PHASE) phases restart from the beginning of the RELOAD phase. However, restart of LOAD RESUME YES or LOAD PART RESUME YES in the BUILD or SORTBLD phase results in message DSNU257I.

Table 41 lists the LOAD phases and their effects on any pending states when the utility is terminated in a particular phase.

Table 41. LOAD phases and their effects on pending states when terminated.

Phase	Effect on pending status
Reload	• Places table space in RECOVER-pending status, then resets the status. • Places indexes in REBUILD-pending status. • Places table space in COPY-pending status. • Places table space in CHECK-pending status.
Build	• Resets REBUILD-pending status for non unique indexes.

Table 41. LOAD phases and their effects on pending states when terminated. (continued)

Phase	Effect on pending status
Indexval	• Resets REBUILD-pending status for unique indexes.
Enforce	• Resets CHECK-pending status for table space.

Restarting LOAD

You can restart LOAD at its last commit point (RESTART(CURRENT)) or at the beginning of the phase during which operation ceased (RESTART(PHASE)). LOAD output messages identify the completed phases; use the DISPLAY command to identify the specific phase during which operation stopped. When using LOAD SHRLEVEL CHANGE, the RESTART(CURRENT) and RESTART(PHASE) operate exactly the same way as committed rows are not loaded again.

By default, DB2 uses RESTART(CURRENT), except if LOAD is restarting during the UTILINIT phase or the UTILTERM phase. In both of these situations, DB2 uses RESTART(PHASE) by default. You can override the default RESTART values by using the RESTART parameter. For general instructions on restarting a utility job, see "Restarting an online utility" on page 43.

The following restrictions apply to restarting LOAD jobs:

- If LOAD abnormally terminates or a system failure occurs while LOAD is in the UTILTERM phase, you must restart with RESTART(PHASE).
- If you restart a LOAD job for a table that has LOB columns that specified the RESUME YES option, you must use RESTART(CURRENT).
- If you use RESTART(PHASE) to restart a LOAD job that specified RESUME NO, the LOB table spaces and indexes on auxiliary tables are reset.
- For a table that has LOB columns, you cannot restart a LOAD job that uses the INCURSOR option.
- If you restart a LOAD job that uses the STATISTICS keyword, inline statistics collection does not occur. To update catalog statistics, run the RUNSTATS utility after the restarted LOAD job completes.
- If you are using a BatchPipes file, you cannot restart the LOAD utility. If the application that populates the BatchPipes file terminates, you need to terminate the job where LOAD is executing. If the LOAD utility was invoked from a stored procedure, you also need to terminate the WLM application environment of the LOAD utility that reads the BatchPipes file. After you terminate the job, terminate the LOAD utility using the DB2 TERM UTILITY command, and then you can resubmit the LOAD job.

Table 42 provides information about restarting LOAD, depending on the phase that LOAD was in when the job stopped. The TYPE column distinguishes between the effects of specifying RESTART or RESTART(PHASE). Additional phase restrictions are explained in the notes.

Table 42. LOAD restart information

Phase	Type of RESTART	Required data sets	Notes
RELOAD	CURRENT	SYSREC and SYSUT1 SYSMAP and SYSERR	1, 2, 10
	PHASE	SYSREC	3, 10
SORT	CURRENT	SYSUT1	4, 10
	PHASE	SYSUT1	10

Table 42. LOAD restart information (continued)

Phase	Type of RESTART	Required data sets	Notes
BUILD	CURRENT	SORTOUT	4, 5, 10
	PHASE	SORTOUT	5, 10
SORTBLD	CURRENT	SYSUT1 and SORTOUT	5, 6, 10
	PHASE	SYSUT1 and SORTOUT	5, 6, 10
INDEXVAL	CURRENT	SYSERR or SYSUT1	2
	PHASE	SYSERR or SYSUT1	2
ENFORCE	CURRENT	SORTOUT and SYSUT1	7
	PHASE	SORTOUT and SYSUT1	7
DISCARD	CURRENT	SYSMAP and SYSERR SORTOUT and SYSUT1	7, 8
	PHASE	SYSMAP and SYSERR SORTOUT and SYSUT1	7, 8
REPORT	CURRENT	SYSERR or SORTOUT SYSMAP and SYSERR	7, 9
	PHASE	SYSERR or SORTOUT SYSMAP and SYSERR	7, 9

Notes:

1. SYSMAP and SYSERR data sets might not be required for all load jobs. See Chapter 16, "LOAD," on page 193 for exact requirements.
2. If the SYSERR data set is not required and has not been provided, LOAD uses SYSUT1 as a work data set to contain error information.
3. You must not restart during the RELOAD phase if you specified SYSREC DD *.This statement prevents internal commits from being taken, and RESTART performs like RESTART(PHASE), except with no data back out. Also, you must not restart if your SYSREC input consistsof multiple, concatenated data sets.
4. The utility can be restarted with either RESTART or RESTART(PHASE). However, because this phase does not take checkpoints, RESTART is always re-executed from the beginning of the phase.
5. A LOAD RESUME YES job cannot be restarted in the BUILD or SORTBLD phase.
6. Use RESTART or RESTART(PHASE) to restart at the beginning of the RELOAD phase.
7. This utility can be restarted with either RESTART or RESTART(PHASE).However, the utility can be re-executed from the last internal checkpoint. This is dependent on the data sets that are used and whether any input data sets have been rewritten.
8. The SYSUT1 data set is required if the target table space is segmented or partitioned.
9. If report is required and this is a load without discard processing, SYSMAP is required to complete the report phase.
10. Any job that finished abnormally in the RELOAD, SORT, BUILD, or SORTBUILD phase restarts from the beginning of the RELOAD phase.

You can restart LOAD at its last commit point or at the beginning of the phase during which operation ceased. LOAD output messages identify the completed phases; use the DISPLAY command to identify the specific phase during which operation stopped.

Restarting after an out-of-space condition: See "Restarting after the output data set is full" on page 45 for guidance in restarting LOAD from the last commit point after receiving an out-of-space condition.

Concurrency and compatibility for LOAD

DB2 treats Individual data and index partitions as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible.

For nonpartitioned secondary indexes, LOAD PART:
- Drains only the logical partition
- Does not set the page set REBUILD-pending status (PSRBD)
- Does not consider PCTFREE or FREEPAGE attributes when inserting keys

Claims and drains: Table 43 shows which claim classes LOAD drains and the restrictive states the utility sets.

Table 43. Claim classes of LOAD operations

Target	LOAD SHRLEVEL NONE	LOAD PART SHRLEVEL NONE	LOAD SHRLEVEL CHANGE	LOAD PART SHRLEVEL CHANGE
Table space, index, or physical partition of a table space or index space	DA/UTUT	DA/UTUT	CW/UTRW	CW/UTRW
Nonpartitioned secondary index	DA/UTUT	DR	CW/UTRW	CW/UTRW
Data-partitioned secondary index	DA/UTUT	DA/UTUT	CW/UTRW	CW/UTRW
Index logical partition	None	DA/UTUT	None	CW/UTRW
Primary index (with ENFORCE option only)	DW/UTRO	DW/UTRO	CR/UTRW	CR/UTRW
RI dependents	CHKP (NO)	CHKP (NO)	CHKP (NO)	CHKP (NO)

Legend:
- CHKP (NO): Concurrently running applications do not see CHECK-pending status after commit.
- CR: Claim the read claim class.
- CW: Claim the write claim class.
- DA: Drain all claim classes, no concurrent SQL access.
- DR: Drain the repeatable read class, no concurrent access for SQL repeatable readers.
- DW: Drain the write claim class, concurrent access for SQL readers.
- UTUT: Utility restrictive state, exclusive control.
- UTRO: Utility restrictive state, read-only access allowed.
- UTRW: Utility restrictive state, read-write access allowed.
- None: Object is not affected by this utility.
- RI: Referential integrity

Compatibility: Table 44 shows whether or not utilities are compatible with LOAD and can run concurrently on the same target object. The target object can be a table space, an index space, or a partition of a table space or index space.

Table 44. Compatibility of LOAD with other utilities

Action	LOAD SHRLEVEL NONE	LOAD SHRLEVEL CHANGE
BACKUP SYSTEM	YES	YES
CHECK DATA DELETE NO	No	No
CHECK DATA DELETE YES	No	No

Table 44. Compatibility of LOAD with other utilities (continued)

Action	LOAD SHRLEVEL NONE	LOAD SHRLEVEL CHANGE
CHECK INDEX	No	No
CHECK LOB	No	No
COPY INDEXSPACE SHRLEVEL CHANGE	No	Yes
COPY INDEXSPACE SHRLEVEL REFERENCE	No	No
COPY TABLESPACE SHRLEVEL CHANGE	No	Yes
COPY TABLESPACE SHRLEVEL REFERENCE	No	No
COPYTOCOPY	No	Yes
DIAGNOSE	Yes	Yes
LOAD SHRLEVEL CHANGE	No	Yes
LOAD SHRLEVEL NONE	No	No
MERGECOPY	No	Yes
MODIFY RECOVERY	No	Yes
MODIFY STATISTICS	No	Yes
QUIESCE	No	No
REBUILD INDEX	No	No
RECOVER (no options)	No	No
RECOVER ERROR RANGE	No	No
RECOVER TOCOPY or TORBA	No	No
REORG INDEX	No	No
REORG TABLESPACE UNLOAD CONTINUE or PAUSE	No	No
REORG TABLESPACE UNLOAD ONLY or EXTERNAL	No	No
REPAIR DUMP or VERIFY	No	No
REPAIR LOCATE KEY or RID DELETE or REPLACE	No	No
REPAIR LOCATE TABLESPACE PAGE REPLACE	No	No
REPORT	Yes	No
RESTORE SYSTEM	No	No
RUNSTATS INDEX SHRLEVEL CHANGE	No	Yes
RUNSTATS INDEX SHRLEVEL REFERENCE	No	No
RUNSTATS TABLESPACE SHRLEVEL CHANGE	No	Yes
RUNSTATS TABLESPACE SHRLEVEL REFERENCE	No	No
STOSPACE	Yes	Yes

Table 44. Compatibility of LOAD with other utilities (continued)

Action	LOAD SHRLEVEL NONE	LOAD SHRLEVEL CHANGE
UNLOAD	No	Yes

SQL operations and other online utilities on the same target partition are incompatible.

After running LOAD

The following tasks are described here:

Copying the loaded table space or partition

If you use LOG YES, consider taking a full image copy of the loaded table space or partition to reduce the processing time of subsequent recovery operations. If you also specify RESUME NO or REPLACE, indicating that this is the first load into the table space, taking two or more full image copies is recommended to enable recovery. Alternatively, take primary and backup inline copies when you do a LOAD REPLACE; full table space or partition image copies that are taken after the LOAD completes are not necessary. However, you might need to take images copies of indexes.

Resetting COPY-pending status

If you load with LOG NO and do not take an inline copy, LOAD places a table space in the COPY-pending status. Immediately after that operation, DB2 cannot recover the table space (although you can, by loading it again). Prepare for recovery, and turn off the restriction, by making a full image copy using SHRLEVEL REFERENCE. (If you end the copy job before it is finished, the table space is still in COPY-pending status.)

You can also remove the restriction by using one of these operations:

- LOAD REPLACE LOG YES
- LOAD REPLACE LOG NO with an inline copy
- REORG LOG YES
- REORG LOG NO with an inline copy
- REPAIR SET with NOCOPYPEND

If you use LOG YES and do not make an image copy of the table space, subsequent recovery operations are possible but take longer than if you had made an image copy.

A table space that is in COPY-pending status can be read without restriction; however, it cannot be updated.

Resetting REBUILD-pending status

LOAD places all the index spaces for a table space in the REBUILD-pending status if you end the job (by using the TERM UTILITY command) before it completes the INDEXVAL phase. DB2 places the table space in RECOVER-pending status if you end the job before the job completes the RELOAD phase.

Resetting the RECOVER-pending status depends on when the utility terminated:

- If the data is intact and you have a full image copy of the affected indexes, you can recover the indexes using the RECOVER INDEX utility. Run the DISPLAY DATABASE command and examine the output. Data is intact when the output indicates that the indexes are in REBUILD-pending status and the table space is not in RECOVER-pending status. If you do not have an image copy available, you must rebuild the entire index by using the REBUILD INDEX utility. However, for partitioning indexes and for secondary indexes that are in REBUILD-pending (RBDP) status, you can use the PART option of REBUILD INDEX to rebuild separate partitions of the index.
- If the data is not intact, you can either load the table again or recover it to a prior point of consistency. Run the DISPLAY DATABASE command and examine the output. The recovery puts the table space into COPY-pending status and places all indexes in REBUILD-pending status.

Resetting the CHECK-pending status

LOAD places a table space in the CHECK-pending status if its referential integrity is in doubt or its check constraints are violated. The intent of the restriction is to encourage the use of the CHECK DATA utility, which locates invalid data and, optionally, removes it. If CHECK DATA removes the invalid data, the remaining data satisfies all check and referential constraints and the CHECK-pending restriction is lifted.

Although CHECK DATA is usually preferred, you can also reset the CHECK-pending status by using any of the following operations:

- Drop tables that contain invalid rows.
- Replace the data in the table space, by using LOAD REPLACE and enforcing check and referential constraints.
- Recover all members of the table space that were set to a prior quiesce point.
- Use REPAIR SET with NOCHECKPEND.

Running CHECK DATA after LOAD REPLACE

Suppose that you choose to replace the contents of the project table by using LOAD REPLACE. While doing that, you let LOAD enforce its referential and table check constraints, so that the project table contains only valid records at the end of the job; it is not in the CHECK-pending status. However, its dependent, the project activity table, is placed in CHECK-pending status: some of its rows might have project numbers that are no longer present in the project table. (If the project table had any other dependents, they also would be in CHECK-pending status.)

You want to run CHECK DATA against the table space that contains the project activity table to reset the status. First, review the review the description of DELETE YES and exception tables. Then, when you run the utility, ensure the availability of all table spaces that contain either parent tables or dependent tables of any table in the table spaces that are being checked.

DELETE YES: This option deletes invalid records and resets the status, but it is **not** the default. Use DELETE NO, the default, to find out quickly how large your problem is; you can choose to correct it by reloading, rather than correcting the current situation.

Exception tables: With DELETE YES, you do not use a discard data set to receive copies of the invalid records; instead, you use another DB2 table called an exception table. This section assumes that you already have an exception table available for every table that is subject to referential or table check constraints. (For instructions on creating them, see "Create exception tables" on page 66.)

If you use DELETE YES, you must name an exception table for every descendent of every table in every table space that is being checked. Deletes that are caused by CHECK DATA are not subject to any of the SQL delete rules; they cascade without restraint to the lowest-level descendent.

If table Y is the exception table for table X, name it with the following clause in the CHECK DATA statement:

```
FOR EXCEPTION IN X USE Y
```

Error and sort data sets

The options ERRDDN, WORKDDN, SORTDEVT, and SORTNUM work in CHECK DATA just as they do in LOAD. That is, you need an error data set, and you can name work data sets for sort and merge processing or let DB2 allocate them dynamically.

Example: In the following example, CHECK DATA is to be run against the table space that contains the project activity table. Assume that the exception tables DSN8810.EPROJACT and DSN8810.EEPA exist.

```
CHECK DATA TABLESPACE DSN8D81A.PROJACT
   DELETE YES
   FOR EXCEPTION IN DSN8810.PROJACT USE DSN8810.EPROJACT
                 IN DSN8810.EMPPROJACT USE DSN8810.EEPA
   SORTDEVT SYSDA
   SORTNUM 4
```

If the statement does not name error or work data sets, the JCL for the job must contain DD statements similar to the following DD statements:

```
//SYSERR   DD UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSUT1   DD UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT DD UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//UTPRINT DD SYSOUT=A
```

Running CHECK DATA after LOAD RESUME

Suppose now that you want to add records to both the project and project activity tables by using LOAD RESUME. Furthermore, you want to run both jobs at the same time, which you can do because the tables belong to separate table spaces. The only new consideration is that you must load the project activity table by using ENFORCE NO because you cannot assume that the parent project table is already fully loaded.

When the two jobs are complete, what table spaces are in CHECK-pending status?

- If you enforced constraints when loading the project table, the table space is **not** in CHECK-pending status.
- Because you did not enforce constraints on the project activity table, the table space **is** in CHECK-pending status.

- Because you used LOAD RESUME (not LOAD REPLACE) when loading the project activity table, its dependents (the employee-to-project-activity table) are **not** in CHECK-pending status. That is, the operation might not delete any parent rows from the project table, and therefore might not violate the referential integrity of its dependent. However if you delete records from PROJACT when checking, you still need an exception table for EMPPROJACT.

Therefore you should check the data in the project activity table.

DB2 records the identifier of the first row of the table that might violate referential or table check constraints. For partitioned table spaces, that identifier is in SYSIBM.SYSTABLEPART; for nonpartitioned table spaces, that identifier is in SYSIBM.SYSTABLES. The SCOPE PENDING option speeds the checking by confining it to just the rows that might be in error.

Example: In the following example, CHECK DATA is to be run against the table space that contains the project activity table after LOAD RESUME:

```
CHECK DATA TABLESPACE DSN8D81A.PROJACT
   SCOPE PENDING
   DELETE YES
   FOR EXCEPTION IN DSN8810.PROJACT USE DSN8810.EPROJACT
                 IN DSN8810.EMPPROJACT USE DSN8810.EEPA
   SORTDEVT SYSDA
   SORTNUM 4
```

As before, the JCL for the job needs DD statements to define the error and sort data sets.

Running CHECK INDEX after loading a table that has indexes

The CHECK INDEX utility tests whether an index is consistent with the data it indexes and issues error messages if it finds an inconsistency. If you have any reason to doubt the accuracy of an index (for example, if the result of an SQL SELECT COUNT statement is inconsistent with RUNSTATS output), run CHECK INDEX. You might also want to run CHECK INDEX after any LOAD operation that shows some abnormal condition in its execution, or even run it periodically to verify the accuracy of important indexes.

To rebuild an index that is inconsistent with its data, use the REBUILD INDEX utility.

Recovering a failed LOAD job

To facilitate recovery in case of failure, DB2 inserts the SYSCOPY record at the beginning of the RELOAD phase if LOG YES is specified in the LOAD control statement. As a result, you can recover the data to a point in time before the LOAD by using RECOVER TORBA.

Reorganizing an auxiliary index after LOAD

Indexes on the auxiliary tables are not built during the BUILD phase. Instead, LOB values are inserted (not loaded) into auxiliary tables during the RELOAD phase as each row is loaded into the base table, and each index on the auxiliary table is updated as part of the insert operation. Because the LOAD utility inserts keys into an auxiliary index, free space within the index might be consumed and index page splits might occur. Consider reorganizing an index on the auxiliary table after LOAD completes to introduce free space into the index for future inserts and loads.

Effects of running LOAD

This section contains information about the effects of running the LOAD utility.

The effect of LOAD on index version numbers

DB2 stores the range of used index version numbers in the OLDEST_VERSION and CURRENT_VERSION columns of the following catalog tables:

- SYSIBM.SYSINDEXES
- SYSIBM.SYSINDEXPART

The OLDEST_VERSION column contains the oldest used version number, and the CURRENT_VERSION column contains the current version number.

When you run LOAD with the REPLACE option, the utility updates this range of used version numbers for indexes that are defined with the COPY NO attribute. LOAD REPLACE sets the OLDEST_VERSION column to the current version number, which indicates that only one version is active; DB2 can then reuse all of the other version numbers.

Recycling of version numbers is required when all of the version numbers are being used. All version numbers are being used when one of the following situations is true:

- The value in the CURRENT_VERSION column is one less than the value in the OLDEST_VERSION column.
- The value in the CURRENT_VERSION column is 15, and the value in the OLDEST_VERSION column is 0 or 1.

You can also run REBUILD INDEX, REORG INDEX, or REORG TABLESPACE to recycle version numbers for indexes that are defined with the COPY NO attribute. To recycle version numbers for indexes that are defined with the COPY YES attribute or for table spaces, run MODIFY RECOVERY.

For more information about versions and how they are used by DB2, see Part 2 of *DB2 Administration Guide*.

The effect of LOAD REPLACE on the control interval

When you run a LOAD job with the REPLACE option but without the REUSE option and the data set that contains the data is DB2-managed, DB2 deletes this data set before the LOAD and redefines a new data set with a control interval that matches the page size.

The effect of LOAD on SYSIBM.SYSCOPY

1. LOAD REPLACE LOG(YES) inserts a ICTYPE=R record into SYSCOPY
2. LOAD REPLACE LOG(NO) inserts a ICTYPE=S record into SYSCOPY
3. LOAD LOG(NO)[1] inserts a ICTYPE=Y record into SYSCOPY
4. LOAD LOG(YES)[1] inserts a ICTYPE=Z record into SYSCOPY

Note:

1. For LOAD RESUME YES, if the SYSREC data set is empty, no record will be inserted into the SYSCOPY table.

Sample LOAD control statements

Example 1: Specifying field positions. The LOAD control statement in Figure 39 specifies that the LOAD utility is to load the records from the data set that is defined by the SYSREC DD statement into table DSN8810.DEPT. SYSREC is the default input data set.

Each POSITION clause specifies the location of a field in the input record. In this example, LOAD accepts the input that is shown in Figure 40 and interprets it as follows:

- The first 3 bytes of each record are loaded into the DEPTNO column of the table.
- The next 36 bytes, including trailing blanks, are loaded into the DEPTNAME column.

  If this input column were defined as VARCHAR(36), the input data would need to contain a 2-byte binary length field preceding the data. This binary field would begin at position 4.
- The next three fields are loaded into columns that are defined as CHAR(6), CHAR(3), and CHAR(16).

The RESUME YES clause specifies that the table space does not need to be empty; new records are added to the end of the table.

```
LOAD DATA
 RESUME YES
 INTO TABLE DSN8810.DEPT
 (DEPTNO    POSITION (1:3)     CHAR(3),
    DEPTNAME POSITION (4:39)    CHAR(36),
    MGRNO     POSITION (40:45)   CHAR(6),
    ADMRDEPT POSITION (46:48)   CHAR(3),
    LOCATION POSITION (49:64)   CHAR(16))
```

Figure 39. Example of a LOAD statement that specifies field positions

Figure 40. shows the input to the preceding LOAD job.

```
A00SPIFFY COMPUTER SERVICE DIV.        000010A00USIBMSTODB21
B01PLANNING                            000020A00USIBMSTODB21
C01INFORMATION CENTER                  000030A00USIBMSTODB21
D01DEVELOPMENT CENTER                        A00USIBMSTODB21
```

Figure 40. Records in an input data set for LOAD

Table 45 shows the result of executing the statement SELECT * FROM DSN8810.DEPT after the preceding input records are loaded.

Table 45. Data that is loaded into a table

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	USIBMSTODB21
B01	PLANNING	000020	A00	USIBMSTODB21
C01	INFORMATION CENTER	000030	A00	USIBMSTODB21
D01	DEVELOPMENT CENTER		A00	USIBMSTODB21

Example 2: Replacing data in a given partition. The following control statement specifies that data from the data set that is defined by the SYSREC DD statement is to be loaded into the first partition of table DSN8810.DEPT. The default input data set is SYSREC. The REPLACE option indicates that the input data is to replace only the specified partition. If the REPLACE option was specified before the PART option, REPLACE would indicate that entire table space is to be replaced, and the data is to be loaded into the specified partition. Note that the keyword DATA does not need to be specified.

```
LOAD
  INTO TABLE DSN8810.DEPT PART 1 REPLACE
```

Example 3: Loading selected records into multiple tables. The control statement in Figure 41 specifies that the LOAD utility is to load certain data from the EMPLDS input data set into tables DSN8810.EMP, SMITH.EMPEMPL, and DSN8810.DEPT. The input data set is identified by the INDDN option. The WHEN clauses indicate which records are to be loaded into each table. For the EMP and DEPT tables, the utility is to load only records that begin with the string LKA. For the EMPEMPL table, the utility is to load only records that begin with the string ABC. The RESUME YES option indicates that the table space does not need to be empty for the LOAD job to proceed. The new rows are added to the end of the tables. This example assumes that the first two tables being loaded have exactly the same forma, and that the input data matches that format; therefore, no field specifications are needed for those two INTO TABLE clauses. The third table has a different format, so field specifications are required and are supplied in the example.

The POSITION clauses specify the location of the fields in the input data for the DEPT table. For each source record that is to be loaded into the DEPT table:

- The characters in positions 7 through 9 are loaded into the DEPTNO column.
- The characters in positions 10 through 35 are loaded into the DEPTNAME column.
- The characters in positions 36 through 41 are loaded into the MGRNO column.
- The characters in positions 42 through 44 are loaded into the ADMRDEPT column.

```
LOAD DATA INDDN EMPLDS
 RESUME YES
 INTO TABLE DSN8810.EMP
 WHEN (1:3)='LKA'
 INTO TABLE SMITH.EMPEMPL
 WHEN (1:3)='ABC'
 INTO TABLE DSN8810.DEPT
 WHEN (1:3)='LKA'
 (DEPTNO POSITION (7:9) CHAR,
   DEPTNAME POSITION (10:35) CHAR,
   MGRNO POSITION (36:41) CHAR,
   ADMRDEPT POSITION (42:44) CHAR)
```

Figure 41. Example LOAD statement that loads selected records into multiple tables

Example 4: Loading data of different data types. The control statement in Figure 42 on page 276 specifies that LOAD is to load data from the SYSRECPJ input data set into table DSN8810.PROJ. The input data set is identified by the INDDN option. Assume that the table space that contains table DSN8810.PROJ is currently empty.

For each input record, data is loaded into the specified columns (that is, PROJNO, PROJNAME, DEPTNO, and so on) to form a table row. Any other PROJ columns that are not specified in the LOAD control statement are set to the default value.

The POSITION clauses define the starting positions of the fields in the input data set. The ending positions of the fields in the input data set are implicitly defined either by the length specification of the data type (CHAR *length*) or the length specification of the external numeric data type (LENGTH).

The numeric data that is represented in SQL constant format (EXTERNAL format) is converted to the correct internal format by the LOAD process and placed in the indicated column names. The two dates (PRSTDATE and PRENDATE) are assumed to be represented by eight digits and two separator characters, as in the USA format (for example, 11/15/2001). The length of the date fields is given as 10 explicitly, although in many cases, the default is the same value.

```
LOAD DATA INDDN(SYSRECPJ)
 INTO TABLE DSN8810.PROJ
  (PROJNO   POSITION  (1) CHAR(6),
   PROJNAME POSITION  (8) CHAR(22),
   DEPTNO   POSITION (31) CHAR(3),
   RESPEMP  POSITION (35) CHAR(6),
   PRSTAFF  POSITION (42) DECIMAL EXTERNAL(5),
   PRSTDATE POSITION (48) DATE EXTERNAL(10),
   PRENDATE POSITION (59) DATE EXTERNAL(10),
   MAJPROJ  POSITION (70) CHAR(6))
```

Figure 42. Example of loading data of different data types

Example 5: Loading data in delimited file format. The control statement in Figure 43 on page 277 specifies that data in delimited format is to be loaded into the specified columns (FILENO, DATE1, TIME1, and TIMESTMP) in table TBQB0103. The FORMAT DELIMITED option indicates that the data is in delimited format. The data is to be loaded from the SYSREC data set, which is the default.

The COLDEL option indicates that the column delimiter is a comma (,). The CHARDEL option indicates that the character string delimiter is a double quotation mark ("). The DECPT option indicates that the decimal point character is a period (.). You are not required to explicitly specify these particular characters, because they are all defaults.

```
//*
//STEP3    EXEC DSNUPROC,UID='JUQBU101.LOAD2',TIME=1440,
//         UTPROC='',
//         SYSTEM='SSTR',DB2LEV=DB2A
//SYSERR   DD DSN=JUQBU101.LOAD2.STEP3.SYSERR,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//SYSDISC  DD DSN=JUQBU101.LOAD2.STEP3.SYSDISC,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//SYSMAP   DD DSN=JUQBU101.LOAD2.STEP3.SYSMAP,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//SYSUT1   DD DSN=JUQBU101.LOAD2.STEP3.SYSUT1,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//UTPRINT  DD SYSOUT=*
//SORTOUT  DD DSN=JUQBU101.LOAD2.STEP3.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4096,(20,20),,,ROUND)
//SYSIN    DD *
   LOAD DATA
        FORMAT DELIMITED COLDEL ',' CHARDEL '"' DECPT '.'
        INTO TABLE TBQB0103
            (FILENO   CHAR,
             DATE1    DATE EXTERNAL,
             TIME1    TIME EXTERNAL,
             TIMESTMP TIMESTAMP EXTERNAL)
 /*
//SYSREC   DD *
 "001", 2000-02-16, 00.00.00, 2000-02-16-00.00.00.0000
 "002", 2001-04-17, 06.30.00, 2001-04-17-06.30.00.2000
 "003", 2002-06-18, 12.30.59, 2002-06-18-12.30.59.4000
 "004", 1991-08-19, 18.59.30, 1991-08-19-18.59.30.8000
 "005", 2000-12-20, 24.00.00, 2000-12-20-24.00.00.0000
 /*
```

Figure 43. Example of loading data in delimited file format

Example 6: Concatenating multiple input records. The control statement in Figure 44 on page 278 specifies that data from the SYSRECOV input data set is to be loaded into table DSN8810.TOPTVAL. The input data set is identified by the INDDN option. The table space that contains the TOPTVAL table is currently empty.

Some of the data that is to be loaded into a single row spans more than one input record. In this situation, an X in column 72 indicates that the input record contains fields that are to be loaded into the same row as the fields in the next input record. In the LOAD control statement, `CONTINUEIF(72:72)='X'` indicates that LOAD is to concatenate any input records that have an X in column 72 with the next record before loading the data.

For each assembled input record (that is, after the concatenation), fields are loaded into the DSN8810.TOPTVAL table columns (that is, MAJSYS, ACTION, OBJECT ..., DSPINDEX) to form a table row. Any columns that are not specified in the LOAD control statement are set to the default value.

The POSITION clauses define the starting positions of the fields in the assembled input records. Starting positions are numbered from the first column of the internally assembled input record, not from the start of the input records in the

sequential data set. The ending positions of the fields are implicitly defined by the length specification of the data type (CHAR *length*).

No conversions are required to load the input character strings into their designated columns, which are also defined to be fixed-length character strings. However, because columns INFOTXT, HELPTXT, and PFKTXT are defined as 79 characters in length and the strings that are being loaded are 71 characters in length, those strings are padded with blanks as they are loaded.

```
LOAD DATA INDDN(SYSRECOV) CONTINUEIF(72:72)='X'
  INTO TABLE DSN8810.TOPTVAL
  (MAJSYS   POSITION   (2) CHAR(1),
   ACTION   POSITION   (4) CHAR(1),
   OBJECT   POSITION   (6) CHAR(2),
   SRCHCRIT POSITION   (9) CHAR(2),
   SCRTYPE  POSITION  (12) CHAR(1),
   HEADTXT  POSITION  (80) CHAR(50),
   SELTXT   POSITION (159) CHAR(50),
   INFOTXT  POSITION (238) CHAR(71),
   HELPTXT  POSITION (317) CHAR(71),
   PFKTXT   POSITION (396) CHAR(71),
   DSPINDEX POSITION (475) CHAR(2))
```

Figure 44. Example of concatenating multiple input records before loading the data

Example 7: Loading null values. The control statement in Figure 45 specifies that data from the SYSRECST data set is to be loaded into the specified columns in table SYSIBM.SYSSTRINGS. The input data set is identified by the INDDN option. The NULLIF option for the ERRORBYTE and SUBBYTE columns specifies that if the input field contains a blank, LOAD is to place a null value in the indicated column for that particular row. The DEFAULTIF option for the TRANSTAB column indicates that the utility is to load the default value for this column if the input field value is GG. The CONTINUEIF option indicates that LOAD is to concatenate any input records that have an X in column 80 with the next record before loading the data.

```
LOAD DATA INDDN(SYSRECST) CONTINUEIF(80:80)='X' RESUME(YES)
     INTO TABLE SYSIBM.SYSSTRINGS
           (INCCSID   POSITION(  1) INTEGER EXTERNAL(5),
            OUTCCSID  POSITION(  7) INTEGER EXTERNAL(5),
            TRANSTYPE POSITION( 13) CHAR(2),
            ERRORBYTE POSITION( 16) CHAR(1) NULLIF(ERRORBYTE=' '),
            SUBBYTE   POSITION( 18) CHAR(1) NULLIF(SUBBYTE=' '),
            TRANSPROC POSITION( 20) CHAR(8),
            IBMREQD   POSITION( 29) CHAR(1),
            TRANSTAB  POSITION( 31) CHAR(256) DEFAULTIF(TRANSTYPE='GG'))
```

Figure 45. Example of loading null values

Example 8: Enforcing referential constraints when loading data. The control statement in Figure 46 on page 279 specifies that data from the SYSREC input data set is to be loaded into table DSN8810.PROJ. The default input data set is SYSREC. The table space that contains the PROJ table is not empty. RESUME YES indicates that the records are to be added to the end of the table.

The ENFORCE CONSTRAINTS option indicates that LOAD is to enforce referential constraints on the data that is being added. This option is also the default. All violations are reported in the output. All records causing these violations are not loaded and placed in the SYSDISC data set, which is the default data set for discarded records.

The CONTINUEIF option indicates that before loading the data LOAD is to concatenate any input records that have an X in column 72 with the next record.

```
LOAD DATA INDDN(SYSREC) CONTINUEIF(72:72)='X'
 RESUME YES
 ENFORCE CONSTRAINTS
 INTO TABLE DSN8810.PROJ
  (PROJNO POSITION (1) CHAR (6),
   PROJNAME POSITION (8) VARCHAR,
   DEPTNO POSITION (33) CHAR (3),
   RESPEMP POSITION (37) CHAR (6),
   PRSTAFF POSITION (44) DECIMAL EXTERNAL (5),
   PRSTDATE POSITION (50) DATE EXTERNAL,
   PRENDATE POSITION (61) DATE EXTERNAL,
   MAJPROJ POSITION (80) CHAR (6) NULLIF(MAJPROJ='      '))
```

Figure 46. Example of enforcing referential constraints when loading data

Example 9: Loading data without enforcing referential constraints. The control statement in Figure 47 on page 280 specifies that data from the SYSRECAC input data set is to be loaded into table DSN8810.ACT. The INDDN option identifies the input data set.

ENFORCE NO indicates that the LOAD utility is not to enforce referential constraints and places the table in CHECK-pending status. Use this option if you are loading data into several tables that are related in such a way that the referential constraints cannot be checked until all tables are loaded. For example, a column in table A depends on a column in table B; a column in table B depends on a column in table C; and a column in table C depends on a column in table A.

The POSITION clauses define the starting positions of the fields in the input data set. The ending positions of the fields in the input data set are implicitly defined by the length specification of the data type (CHAR *length*). In this case, the characters in positions 1 through 3 are loaded into the ACTNO column, the characters in positions 5 through 10 are loaded into the ACTKWD column, and the characters in position 13 onward are loaded into the ACTDESC column. Because the ACTDESC column is of type VARCHAR, the input data needs to contain a 2-byte binary field that contains the length of the character field. This binary field begins at position 13.

```
//STEP1    EXEC DSNUPROC,UID='IUIQU2UB.LOAD',
//         UTPROC='',
//         SYSTEM='DSN'
//SYSRECAC   DD DSN=IUIQU2UB.LOAD.DATA,DISP=SHR,VOL=SER=SCR03,
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSUT1   DD DSN=IUIQU2UB.LOAD.STEP1.SYSUT1,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT  DD DSN=IUIQU2UB.LOAD.STEP1.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
LOAD DATA INDDN(SYSRECAC) RESUME YES
     INTO TABLE DSN8810.ACT
          (ACTNO    POSITION(1) INTEGER EXTERNAL(3),
           ACTKWD   POSITION(5) CHAR(6),
           ACTDESC POSITION(13) VARCHAR)
     ENFORCE NO
//*
```

Figure 47. Example of loading data without enforcing referential constraints

Example 10: Loading data by using a parallel index build. The control statement in Figure 48 on page 281 specifies that data from the SYSREC input data set is to be loaded into table DSN8810.DEPT. Assume that 22 000 rows need to be loaded into table DSN8810.DEPT, which has three indexes. In this example, the SORTKEYS option is used to improve performance by forcing a parallel index build. The SORTKEYS option specifies 66 000 as an estimate of the number keys to sort in parallel during the SORTBLD phase. (This estimate was computed by using the calculation that is described in "Improved performance with SORTKEYS" on page 253.) Because more than one index needs to be built, LOAD builds the indexes in parallel.

The SORTDEVT and SORTNUM keywords specify that DFSORT is to dynamically allocate the required data sets. If sufficient virtual storage resources are available, one utility subtask pair is started to build each index. This example does not require UTPRIN*nn* DD statements because it uses DSNUPROC to invoke utility processing, which includes a DD statement that allocates UTPRINT to SYSOUT.

The CONTINUEIF option indicates that, before loading the data, LOAD is to concatenate any input records that have a plus sign (+) in column 79 and a plus sign (+) in column 80 with the next record.

```
//SAMPJOB  JOB  ...
//STEP1    EXEC DSNUPROC,UID='SAMPJOB.LOAD',UTPROC='',SYSTEM='DSN'
//SORTOUT  DD DSN=SAMPJOB.LOAD.STEP1.SORTOUT,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SYSUT1   DD DSN=SAMPJOB.LOAD.STEP1.SYSUT1,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SYSERR   DD DSN=SAMPJOB.LOAD.STEP1.SYSERR,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(2000,(20,20),,,ROUND)
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400)
//SYSMAP   DD DSN=SAMPJOB.LOAD.STEP1.SYSMAP,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(2000,(20,20),,,ROUND),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400)
//SYSREC   DSN=SAMPJOB.TEMP.DATA,DISP=SHR,UNIT=SYSDA
//SYSIN    DD *
LOAD DATA REPLACE INDDN SYSREC CONTINUEIF(79:80)='++'
 SORTKEYS 66000 SORTDEVT SYSDA SORTNUM 3
 INTO TABLE DSN8810.DEPT
/*
```

Figure 48. Example of loading data by using a parallel index build

Example 11: Creating inline copies. The LOAD control statement in Figure 49 on page 282 specifies that the LOAD utility is to load data from the SYSREC data set into the specified columns of table ADMF001.TB0S3902. See "Example 1: Specifying field positions" on page 274. for an explanation of the POSITION clauses.

COPYDDN(COPYT1) indicates that LOAD is to create inline copies and write the primary image copy to the data set that is defined by the COPYT1 template. This template is defined in one of the preceding TEMPLATE control statements. For more information about TEMPLATE control statements, see "Syntax and options of the TEMPLATE control statement " on page 593 of the TEMPLATE chapter. To create an inline copy, you must also specify the REPLACE option, which indicates that any data in the table space is to be replaced.

`CONTINUEIF(79:80)='++'` indicates that, before loading the data, LOAD is to concatenate any input records that have a plus sign (+) in column 79 and a plus sign (+) in column 80 with the next record.

The ERRDDN(ERRDDN) and MAPDDN(MAP) options indicate that information about errors is to be written to the data sets that are defined by the ERRDDN and MAP templates. DISCARDDDN(DISCARD) specifies that discarded records (those that violate referential constraints) are to be written to the data set that is defined by the DISCARD template. WORKDDN(UT1,OUT) specifies the temporary work files for sort input and output; LOAD is to use the data set that is defined by the UT1 template for sort input and the data set that is defined by the OUT template for sort output.

```
//STEP1    EXEC DSNUPROC,UID='JUOSU339.LOAD1',TIME=1440,
//         UTPROC='',
//         SYSTEM='SSTR',DB2LEV=DB2A
//SYSREC   DD DSN=CUST.FM.CINT135.DATA,DISP=SHR,VOL=SER=FORDMD,
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
  TEMPLATE ERRDDN  UNIT(SYSDA)
                   DSN(JUOSU339.T&TI..&ST..ERRDDN)
                   SPACE(50,10) TRK
  TEMPLATE UT1     UNIT(SYSDA)
                   DSN(JUOSU339.T&TI..&ST..SYSUT1)
                   SPACE(50,10) TRK
  TEMPLATE OUT     UNIT(SYSDA)
                   DSN(JUOSU339.T&TI..&ST..SYSOUT)
                   SPACE(50,10) TRK
  TEMPLATE MAP     UNIT(SYSDA)
                   DSN(JUOSU339.T&TI..&ST..SYSMAP)
                   SPACE(50,10) TRK
  TEMPLATE DISCARD UNIT(SYSDA)
                   DSN(JUOSU339.T&TI..&ST..DISCARD)
                   SPACE(50,10) TRK
  TEMPLATE COPYT1
           UNIT(SYSDA)
           DSN(JUOSU339.COPY1.STEP1.&SN..COPY&LR.&PB.)
           DISP(MOD,CATLG,CATLG)
           SPACE(60,30) TRK
  LOAD DATA INDDN SYSREC REPLACE
   CONTINUEIF(79:80)='++'
   COPYDDN(COPYT1)
   ERRDDN(ERRDDN)
   WORKDDN(UT1,OUT)
   MAPDDN(MAP)
   DISCARDDN(DISCARD)
  INTO TABLE
       ADMF001.TBOS3902
  ( ID_PARTITION       POSITION(1)   CHAR(1),
    CD_PLANT           POSITION(2)   CHAR(5),
    NO_PART_BASE       POSITION(7)   CHAR(9),
    NO_PART_PREFIX     POSITION(16)  CHAR(7),
   NO_PART_SUFFIX     POSITION(23)  CHAR(8),
  NO_PART_CONTROL    POSITION(31)   CHAR(3),
  DT_TRANS_EFFECTIVE POSITION(34)   DATE EXTERNAL(10),
  CD_INV_TRANSACTION POSITION(44)   CHAR(3),
  TS_PROCESS         POSITION(47)   TIMESTAMP EXTERNAL(26),
  QT_INV_TRANSACTION POSITION(73)   INTEGER,
  CD_UNIT_MEAS_USAGE POSITION(77)   CHAR(2),
  CD_USER_ID         POSITION(79)   CHAR(7),
  NO_DEPT            POSITION(86)   CHAR(4),
  NO_WORK_CENTER     POSITION(90)   CHAR(6))
/*
```

Figure 49. Example of creating inline copies

Example 12: Collecting statistics. The example in Figure 50 on page 284 is similar to example 11, except that the STATISTICS option and other related options have been added so that during the LOAD job, DB2 also gathers statistics for the table space. Gathering these statistics eliminates the need to run the RUNSTATS utility after completing the LOAD operation.

The TABLE, COLUMN, and INDEX options specify that information is to be gathered for columns QT_INV_TRANSACTION, NO_DEPT, NO_PART_PREFIX, DT_TRANS_EFFECTIVE and index ID0S3902 for table TB0S3902. SAMPLE 53 indicates that LOAD is to sample 53% of the rows when gathering non-indexed column statistics. For the index, the KEYCARD option specifies that all of the distinct values in all of the key column combinations are to be collected. FREQVAL

NUMCOLS 4 COUNT 20 indicates that 20 frequent values are to be collected on the concatenation of the first four key columns.

REPORT YES indicates that the statistics are to be sent to SYSPRINT as output. UPDATE ALL and HISTORY ALL indicate that all collected statistics are to be updated in the catalog and catalog history tables.

```
//STEP1    EXEC DSNUPROC,UID='JUOSU339.LOAD1',TIME=1440,
//         UTPROC='',
//         SYSTEM='SSTR',DB2LEV=DB2A
//SYSREC   DD DSN=CUST.FM.CINT135.DATA,DISP=SHR,VOL=SER=FORDMD,
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
  TEMPLATE ERRDDN  UNIT(SYSDA)
                   DSN(JUOSU339.T&TI..&ST..ERRDDN)
                   SPACE(50,10) TRK
  TEMPLATE UT1     UNIT(SYSDA)
                   DSN(JUOSU339.T&TI..&ST..SYSUT1)
                   SPACE(50,10) TRK
  TEMPLATE OUT     UNIT(SYSDA)
                   DSN(JUOSU339.T&TI..&ST..SYSOUT)
                   SPACE(50,10) TRK
  TEMPLATE MAP     UNIT(SYSDA)
                   DSN(JUOSU339.T&TI..&ST..SYSMAP)
                   SPACE(50,10) TRK
  TEMPLATE DISCARD UNIT(SYSDA)
                   DSN(JUOSU339.T&TI..&ST..DISCARD)
                   SPACE(50,10) TRK
  TEMPLATE COPYT1
           UNIT(SYSDA)
           DSN(JUOSU339.COPY1.STEP1.&SN..COPY&LR.&PB.)
           DISP(MOD,CATLG,CATLG)
           SPACE(60,30) TRK
  LOAD DATA INDDN SYSREC REPLACE
   CONTINUEIF(79:80)='++'
   COPYDDN(COPYT1)
   STATISTICS
     TABLE (TBOS3902) SAMPLE 53
       COLUMN (QT_INV_TRANSACTION,
               NO_DEPT,
               NO_PART_PREFIX,
               DT_TRANS_EFFECTIVE)
     INDEX (IDOS3902 KEYCARD
       FREQVAL NUMCOLS 4 COUNT 20)
     REPORT YES UPDATE ALL HISTORY ALL
   ERRDDN(ERRDDN)
   WORKDDN(UT1,OUT)
   MAPDDN(MAP)
   DISCARDDN(DISCARD)
  INTO TABLE
       ADMF001.TBOS3902
  ( ID_PARTITION       POSITION(1)   CHAR(1),
    CD_PLANT           POSITION(2)   CHAR(5),
    NO_PART_BASE       POSITION(7)   CHAR(9),
    NO_PART_PREFIX     POSITION(16)  CHAR(7),
   NO_PART_SUFFIX     POSITION(23)  CHAR(8),
  NO_PART_CONTROL    POSITION(31)   CHAR(3),
  DT_TRANS_EFFECTIVE POSITION(34)   DATE EXTERNAL(10),
  CD_INV_TRANSACTION POSITION(44)   CHAR(3),
  TS_PROCESS         POSITION(47)   TIMESTAMP EXTERNAL(26),
  QT_INV_TRANSACTION POSITION(73)   INTEGER,
  CD_UNIT_MEAS_USAGE POSITION(77)   CHAR(2),
  CD_USER_ID         POSITION(79)   CHAR(7),
  NO_DEPT            POSITION(86)   CHAR(4),
  NO_WORK_CENTER     POSITION(90)   CHAR(6))
/*
```

Figure 50. Example of collecting statistics

Example 13: Loading Unicode data. The following control statement specifies that Unicode data from the REC1 input data set is to be loaded into table ADMF001.TBMG0301. The UNICODE option specifies the type of input data. Only data that satisfies the condition that is specified in the WHEN clause is to be

loaded. The CCSID option specifies the three coded character set identifiers for the input file: one for SBCS data, one for mixed data, and one for DBCS data. LOG YES indicates that logging is to occur during the LOAD job.

```
LOAD DATA INDDN REC1     LOG YES REPLACE
   UNICODE CCSID(00367,01208,01200)
   INTO TABLE "ADMF001 "."TBMG0301"
   WHEN(00004:00005 = X'0003')
```

Example 14: Loading data from multiple input data sets by using partition parallelism. The LOAD control statement in Figure 51 on page 286 contains a series of INTO TABLE statements that specify which data is to be loaded into which partitions of table DBA01.TBLX3303. For each INTO TABLE statement:

- Data is to be loaded into the partition that is identified by the PART option. For example, the first INTO TABLE statement specifies that data is to be loaded into the first partition of table DBA01.TBLX3303.
- Data is to be loaded from the data set that is identified by the INDDN option. For example, the data from the PART1 data set is to be loaded into the first partition.
- Any discarded rows are to be written to the data set that is specified by the DISCARDDN option. For example, rows that are discarded during the loading of data from the PART1 data set are written to the DISC1 data set.
- The data is loaded into the specified columns (EMPNO, LASTNAME, and SALARY).

LOAD uses partition parallelism to load the data into these partitions.

The TEMPLATE utility control statement defines the data set naming convention for the data set that is to be dynamically allocated during the following LOAD job. The name of the template is ERR3. The ERRDDN option in the LOAD statement specifies that any errors are to be written to the data set that is defined by this ERR3 template. For more information about TEMPLATE control statements, see “Syntax and options of the TEMPLATE control statement ” on page 593 in the TEMPLATE chapter.

```
 TEMPLATE ERR3
         DSN &UT..&JO..&ST..ERR3&MO.&DAY.
         UNIT SYSDA  DISP(NEW,CATLG,CATLG)
  LOAD DATA
    REPLACE
    ERRDDN ERR3
    INTO TABLE DBA01.TBLX3303
      PART 1
      INDDN PART1
      DISCARDDN DISC1
        (EMPNO      POSITION(1)    CHAR(6),
         LASTNAME   POSIITON(8)    VARCHAR(15),
         SALARY     POSITION(25)   DECIMAL(9,2))
    .
    .
    .
    INTO TABLE DBA01.TBLX3303
      PART 5
      INDDN PART5
      DISCARDDN DISC5
        (EMPNO      POSITION(1)    CHAR(6),
         LASTNAME   POSIITON(8)    VARCHAR(15),
         SALARY     POSITION(25)   DECIMAL(9,2))
/*
```

Figure 51. Example of loading data from individual data sets

Example 15: Loading data from another table in the same system by using a declared cursor. The following LOAD control statement specifies that all rows that are identified by cursor C1 are to be loaded into table MYEMP. The INCURSOR option is used to specify cursor C1, which is defined in the EXEC SQL utility control statement. Cursor C1 points to the rows that are returned by executing the statement SELECT * FROM DSN8810.EMP. In this example, the column names in table DSN8810.EMP are the same as the column names in table MYEMP. Note that the cursor cannot be defined on the same table into which DB2 is to load the data.

```
EXEC SQL
  DECLARE C1 CURSOR FOR SELECT * FROM DSN8810.EMP
ENDEXEC
LOAD DATA
INCURSOR(C1)
REPLACE
INTO TABLE MYEMP
STATISTICS
```

Example 16: Loading data partitions in parallel from a remote site by using a declared cursor. The LOAD control statement in Figure 52 on page 287 specifies that for each specified partition of table MYEMPP, the rows that are identified by the specified cursor are to be loaded. In each INTO TABLE statement, the PART option specifies the partition number, and the INCURSOR option specifies the cursor. For example, the rows that are identified by cursor C1 are to be loaded into the first partition. The data for each partition is loaded in parallel.

Each cursor is defined in a separate EXEC SQL utility control statement and points to the rows that are returned by executing the specified SELECT statement. These SELECT statement are being executed on a table at a remote server, so the three-part name is used to identify the table. In this example, the column names in table CHICAGO.DSN8810.EMP are the same as the column names in table MYEMPP.

```
EXEC SQL
  DECLARE C1 CURSOR FOR SELECT * FROM CHICAGO.DSN8810.EMP
  WHERE EMPNO <= '099999'
ENDEXEC
EXEC SQL
  DECLARE C2 CURSOR FOR SELECT * FROM CHICAGO.DSN8810.EMP
  WHERE EMPNO > '099999' AND EMPNO <= '199999'
ENDEXEC
EXEC SQL
  DECLARE C3 CURSOR FOR SELECT * FROM CHICAGO.DSN8810.EMP
  WHERE EMPNO > '199999' AND EMPNO <= '299999'
ENDEXEC
EXEC SQL
  DECLARE C4 CURSOR FOR SELECT * FROM CHICAGO.DSN8810.EMP
  WHERE EMPNO > '299999' AND EMPNO <= '999999'
ENDEXEC
LOAD DATA
  INTO TABLE MYEMPP PART 1 REPLACE INCURSOR(C1)
  INTO TABLE MYEMPP PART 2 REPLACE INCURSOR(C2)
  INTO TABLE MYEMPP PART 3 REPLACE INCURSOR(C3)
  INTO TABLE MYEMPP PART 4 REPLACE INCURSOR(C4)
```

Figure 52. Example of loading data partitions in parallel using a declared cursor

Example 17: Loading LOB data from a file The LOAD control statement in Figure 53 specifies that data from 000130DSN!10.SDSNIVPD(DSN8R130) is to be loaded into the MY_EMP_PHOTO_RESUME table. The characters in positions 1 through 6 are loaded into the EMPNO column, and the characters starting from position 7 are to be loaded into the RESUME column. CLOBF indicates that the characters in position 7 are the name of a file from which a CLOB is to be loaded.

REPLACE indicates that the new data will replace any existing data. Although no logging is to be done, as indicated by the LOG NO option, the table space is not to be set in CHECK-pending state, because NOCOPYPEND is specified.

SORTKEYS 1 indicates that one index key is to be sorted.

```
//*****************************************************************
//*   LOAD LOB from file
//*****************************************************************
//LOADIT   EXEC DSNUPROC,UID='LOADIT',TIME=1440,
//         UTPROC='',
//         SYSTEM='DSN'
//SYSREC   DD*
000130DSN!10.SDSNIVPD(DSN8R130)
//SYSUT1   DD DSN=SYSADM.LOAD.SYSUT1,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT  DD DSN=SYSADM.LOAD.SORTOUT,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD  *
 LOAD DATA
   REPLACE LOG NO NOCOPYPEND
   SORTKEYS 1
      INTO TABLE MY_EMP_PHOTO_RESUME
      (EMPNO    POSITION(1:6) CHAR(6),
       RESUME   POSITION(7:31)   CHAR CLOBF)
```

Figure 53. Example of loading LOB data from a file

Chapter 17. MERGECOPY

The MERGECOPY online utility merges image copies that the COPY utility produces, image copies that the COPYTOCOPY utility produces, or inline copies that the LOAD or REORG utilities produce. It can merge several incremental copies of a table space to make one incremental copy. It can also merge incremental copies with a full image copy to make a new full image copy. You cannot run MERGECOPY on concurrent copies.

MERGECOPY operates on the image copy data sets of a table space, and not on the table space itself.

For a diagram of MERGECOPY syntax and a description of available options, see "Syntax and options of the MERGECOPY control statement" on page 290. For detailed guidance on running this utility, see "Instructions for running MERGECOPY" on page 292.

Output: Output from the MERGECOPY utility consists of one of the following types of copies:
- A new single incremental image copy
- A new full image copy

You can create the new image copy for the local or recovery site.

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:
- IMAGCOPY privilege for the database
- DBADM, DBCTRL, or DBMAINT authority for the database
- SYSCTRL or SYSADM authority

An ID with installation SYSOPR authority can also execute MERGECOPY, but only on a table space in the DSNDB01 or DSNDB06 database.

Restrictions on running MERGECOPY:
- MERGECOPY cannot merge image copies into a single incremental image copy for the other site, that is:
  - At local sites, you cannot use RECOVERYDDN with NEWCOPY NO.
  - At recovery sites, you cannot use COPYDDN with NEWCOPY NO.
- When none of the keywords NEWCOPY, COPYDDN, or RECOVERYDDN is specified, the default, NEWCOPY NO COPYDDN(SYSCOPY), is valid for the local site only.
- You cannot run MERGECOPY on concurrent copies.

Execution phases of MERGECOPY:

The MERGECOPY utility operates in these phases:

Phase	Description
UTILINIT	Performs initialization
MERGECOP	Merges incremental copies
UTILTERM	Performs cleanup

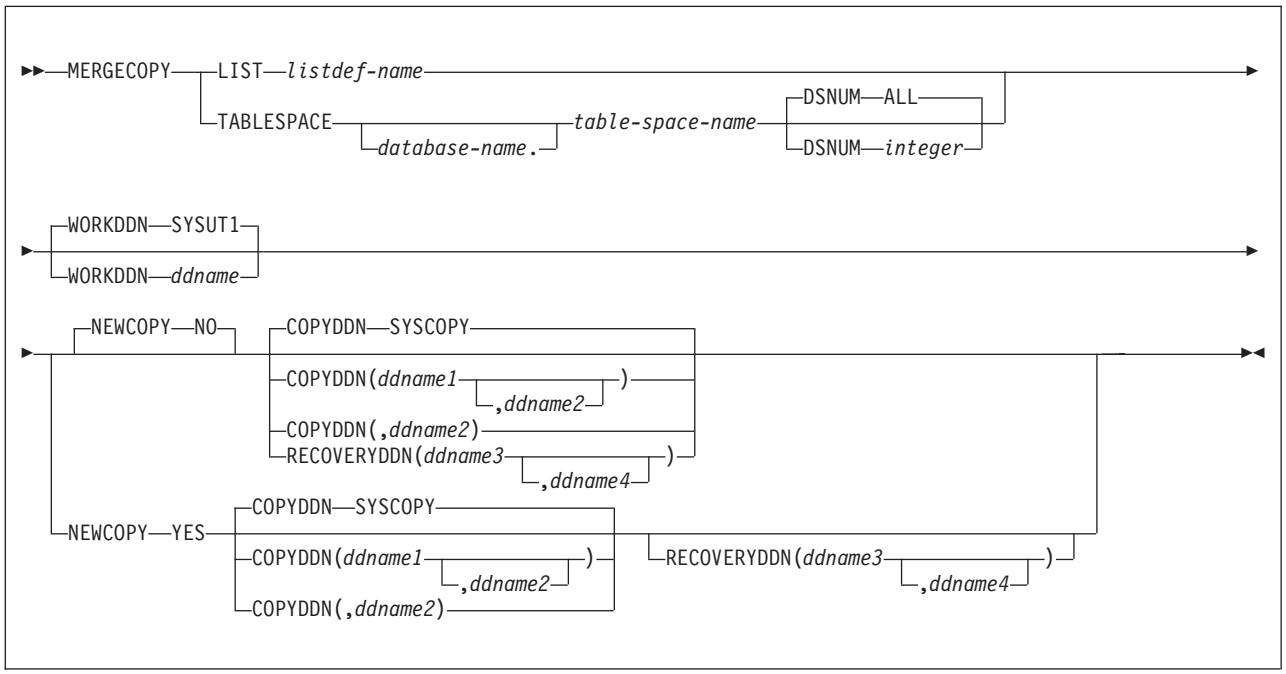
The following topics provide additional information:

- "Syntax and options of the MERGECOPY control statement"
- "Instructions for running MERGECOPY" on page 292
- "Concurrency and compatibility for MERGECOPY" on page 296
- "Sample MERGECOPY control statements" on page 296

Syntax and options of the MERGECOPY control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, you can use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

LIST *listdef-name*

Specifies the name of a previously defined LISTDEF list name that contains only table spaces. You can specify one LIST keyword per MERGECOPY control statement. Do not specify LIST with the TABLESPACE keyword. MERGECOPY is invoked once for each table space in the list. For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

TABLESPACE *database-name.table-space-name*

Specifies the table space that is to be copied, and, optionally, the database to which it belongs.

database-name
The name of the database that the table space belongs to. The **default** is **DSNDB04**.

table-space-name
The name of the table space whose incremental image copies are to be merged.

You cannot run the MERGECOPY utility on the DSNDB01.DBD01, DSNDB01.SYSUTILX, or DSNDB06.SYSCOPY table spaces because you cannot make incremental copies of those table spaces. Because MERGECOPY does not directly access the table space whose copies it is merging, it does not interfere with concurrent access to that table space.

DSNUM
Identifies the table space or a partition or data set within the table space that is to be merged. DSNUM is optional.

ALL Merges the entire table space. The **default** is **ALL**.

integer Is the number of a partition or data set that is to be merged. The maximum is 4096.

For a partitioned table space, the integer is its partition number.

For a nonpartitioned table space, find the integer at the end of the data set name as cataloged in the VSAM catalog. The data set name has the following format, where *y* is either I or J and *nnn* is the data set integer:

catname.DSNDB*x*.*dbname*.*tsname*.*y*0001.A*nnn*

You cannot specify DSNUM and LIST in the same MERGECOPY control statement. Use PARTLEVEL on the LISTDEF instead. If image copies were taken by data set (rather than by table space), MERGECOPY must use the copies by data set.

WORKDDN *ddname*
Specifies a DD statement for a temporary data set or template, which is to be used for intermediate merged output. WORKDDN is optional.

ddname is the DD name. The **default** is **SYSUT1**.

Use the WORKDDN option if you are not able to allocate enough data sets to execute MERGECOPY; in that case, a temporary data set is used to hold intermediate output. If you omit the WORKDDN option, you might find that only some of the image copy data sets are merged. When MERGECOPY has ended, a message is issued that tells the number of data sets that exist and the number of data sets that have been merged. To continue the merge, repeat MERGECOPY with a new output data set.

NEWCOPY
Specifies whether incremental image copies are to be merged with the full image copy. NEWCOPY is optional.

NO
Merges incremental image copies into a single incremental image copy but does not merge them with the full image copy. The **default** is **NO**.

YES
Merges all incremental image copies with the full image copy to form a new full image copy.

COPYDDN (*ddname1,ddname2***)**
Specifies the DD statements for the output image copy data sets at the local site. *ddname1* is the primary output image copy data set. *ddname2* is the backup output image copy data set. COPYDDN is optional.

The **default** is **COPYDDN**(**SYSCOPY**), where SYSCOPY identifies the primary data set.

The COPYDDN keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

RECOVERYDDN (*ddname3,ddname4***)**
Specifies the DD statements for the output image copy data sets at the recovery site. You can have a maximum of two output data sets; the outputs are identical. *ddname3* is the primary output image copy data set. *ddname4* is the backup output image copy data set. RECOVERYDDN is optional. No default value exists for RECOVERYDDN.

The RECOVERYDDN keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

Instructions for running MERGECOPY

To run MERGECOPY, you must:

1. Prepare the necessary data sets, as described in "Data sets that MERGECOPY uses."
2. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for MERGECOPY, see "Sample MERGECOPY control statements" on page 296.)
3. Prepare a utility control statement, specifying the options for the tasks that you want to perform, as described in "Instructions for specific tasks" on page 294.
4. Check the compatibility table in "Concurrency and compatibility for MERGECOPY" on page 296 if you want to run other jobs concurrently on the same target objects.
5. Plan for restart if the MERGECOPY job doesn't complete, as described in "Terminating or restarting MERGECOPY" on page 296.
6. Run MERGECOPY by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Data sets that MERGECOPY uses

Table 46 on page 293 lists the data sets that MERGECOPY uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 46. Data sets that MERGECOPY uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
Image copy data set	Image copy data set that contains the resulting image copy. Specify its DD name with the COPYDDN option of the utility control statement. The default DD name is SYSCOPY.	Yes
Work data set	A temporary data set that is used for intermediate merged output. Specify its DD name with the WORKDDN option of the utility control statement. The default DD name is SYSUT1.	Yes
Input data sets	Image copy data sets that you can preallocate. You define the DD names.	No

The following object is named in the utility control statement and does not require a DD statement in the JCL:

Table space
Object whose copies are to be merged.

Data sets: The input data sets for the merge operation are dynamically allocated. To merge incremental copies, allocate in the JCL a work data set (WORKDDN) and up to two new copy data sets (COPYDDN) for the utility job. You can allocate the data sets to tape or disk. If you allocate them to tape, you need an additional tape drive for each data set.

With the COPYDDN option of MERGECOPY, you can specify the DD names for the output data sets. The option has the format `COPYDDN (ddname1,ddname2)`, where *ddname1* is the DD name for the primary output data set in the system that currently runs DB2, and *ddname2* is the DD name for the backup output data set in the system that currently runs DB2. The default for *ddname1* is SYSCOPY.

The RECOVERYDDN option of MERGECOPY lets you specify the output image copy data sets at the recovery site. The option has the format `RECOVERYDDN (ddname3, ddname4)`, where *ddname3* is the DD name for the primary output image copy data set at the recovery site, and *ddname4* is the DD name for the backup output data set at the recovery site.

Defining the work data set: The work data set should be at least equal in size to the largest input image copy data set that is being merged. Use the same DCB attributes that are used for the image copy data sets.

Creating the control statement

See “Syntax and options of the MERGECOPY control statement” on page 290 for MERGECOPY syntax and option descriptions. See “Sample MERGECOPY control statements” on page 296 for examples of MERGECOPY usage.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in your utility control statement:

Specifying full or incremental image copy

Use the NEWCOPY parameter if the new copy that MERGECOPY creates is to be an incremental image copy or a full image copy. In general, creating a new full image copy is recommended. The reasons for this recommendation are as follows:

- A new full image copy creates a new recovery point.
- The additional time that it takes to create a new full image copy does not have any adverse effect on the access to the table space. The only concurrency implication is the access to SYSIBM.SYSCOPY.
- The range of log records that are to be applied by RECOVER is the same for both the new full image copy and the merged incremental image copy.
- Assuming that the copies are on tape, only one tape drive is required for image copies during a RECOVER.

If NEWCOPY is YES, the utility inserts an entry for the new full image copy into the SYSIBM.SYSCOPY catalog table.

If NEWCOPY is NO, the utility:

- Replaces the SYSCOPY records of the incremental image copies that were merged with an entry for the new incremental image copy
- Deletes all SYSCOPY records of the incremental image copies that have been merged.

In either case, if any of the input data sets might not be allocated, or you did not specify a temporary work data set (WORKDDN), the utility performs a partial merge.

For large table spaces, consider using MERGECOPY to create full image copies.

Use MERGECOPY NEWCOPY YES immediately after each incremental image copy. When you use this option, dates become a valid criterion for deletion of image copy data sets and archive logs. A minimum number of tape drives are allocated for MERGECOPY and RECOVER execution.

Merging online copies

If you merge an online copy with incremental copies, the result is a full inline copy. The data set is logically equivalent to a full image copy, but the data within the data set differs in some respects. See "Using inline COPY with LOAD" on page 252 for additional information about inline copies.

Using MERGECOPY with individual data sets

Use MERGECOPY on copies of an entire table space, on individual data sets, or on partitions. However, MERGECOPY can only merge incremental copies of the same type. That is, you cannot merge incremental copies of an entire table space with

incremental copies of individual data sets to form new incremental copies. The attempt to mix the two types of incremental copies results in the following messages:

```
DSNU460I DSNUBCLO - IMAGE COPIES INCONSISTENT.
                    MERGECOPY REQUEST REJECTED
DSNU010I DSNUGBAC - UTILITY EXECUTION COMPLETE,
                    HIGHEST RETURN CODE=4
```

With the NEWCOPY YES option, however, you can merge a full image copy of a table space with incremental copies of the table space and of individual data sets to make a new full image copy of the table space.

If the image copy data sets that you want to merge reside tape, refer to "Retaining tape mounts" on page 383 for general information about specifying the appropriate parameters on the DD statements.

Using MERGECOPY or COPY

COPY and MERGECOPY can create a full image copy. COPY is required after a LOAD or REORG with LOG NO unless an inline copy is created. However, in other cases an incremental image copy followed by MERGECOPY is a valid alternative.

Avoiding MERGECOPY LOG RBA inconsistencies

MERGECOPY does not use information that was logged between the time of the most recent image copy and the time when MERGECOPY was run. Therefore, you cannot safely delete all log records made before running MERGECOPY. (You can safely delete all log records if you run MODIFY RECOVERY and specify the date when MERGECOPY was run as the value of DATE.)

To delete all log information that is included in a copy that MERGECOPY makes, perform the following steps:

1. Find the record of that copy in the catalog table SYSIBM.SYSCOPY. You can find it by selecting database name, table space name, and date (columns DBNAME, TSNAME, and ICDATE).
2. Column START_RBA contains the RBA of the last image copy that MERGECOPY used. Find the record of the image copy that has the same value of START_RBA.
3. In that record, find the date in column ICDATE. You can use MODIFY RECOVERY to delete all copies and log records for the table space that were made before that date.

RECOVER uses the LOG RBA of image copies to determine the starting point in the log that is needed for recovery. Normally, a timestamp directly corresponds to a LOG RBA. Because of this, and because MODIFY uses dates to clean up recovery history, you might decide to use dates to delete old archive log tapes. This decision might cause a problem if you use MERGECOPY. MERGECOPY inserts the LOG RBA of the last incremental image copy into the SYSCOPY row that is created for the new image copy. The date that is recorded in the ICDATE column of SYSCOPY row is the date that MERGECOPY was executed.

Creating image copies in a JES3 environment

Ensure that sufficient units are available to mount the required image copies. In a JES3 environment, if the number of image copies that are to be restored exceeds the number of available online and offline units, and if the MERGECOPY job successfully allocates all available units, the job waits for more units to become available.

Running MERGECOPY on the directory

You cannot run the MERGECOPY utility on the DSNDB01.DBD01, DSNDB01.SYSUTILX, or DSNDB06.SYSCOPY table spaces because you cannot make incremental copies of those table spaces.

Terminating or restarting MERGECOPY

For instructions on terminating a utility job, see "Terminating an online utility with the TERM UTILITY command" on page 42.

By default, MERGECOPY restarts at the beginning of the current phase. For instructions on restarting a utility job, see "Restarting an online utility" on page 43.

See "Restarting after the output data set is full" on page 45 for guidance in restarting MERGECOPY from the last commit point after receiving an out-of-space condition.

Concurrency and compatibility for MERGECOPY

DB2 treats individual data and index partitions as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible.

Table 47 shows the restrictive state that the utility sets on the target object.

Table 47. Claim classes of MERGECOPY operations.

Target	MERGECOPY
Table space or partition	UTRW

Legend:

UTRW - Utility restrictive state - read-write access allowed.

MERGECOPY can run concurrently on the same target object with any utility **except the following utilities**:

- COPY TABLESPACE
- LOAD
- MERGECOPY
- MODIFY
- RECOVER
- REORG TABLESPACE
- UNLOAD (only when from the same image copy data set)

The target object can be a table space or partition.

Sample MERGECOPY control statements

Example 1: Creating a merged incremental copy. The control statement in Figure 54 on page 297 specifies that the MERGECOPY utility is to merge incremental image copies from table space DSN8S81C into a single incremental image copy. The NEWCOPY NO option indicates that these incremental copies are not to be merged with the full image copy. The COPYDDN option specifies that the output image copies are to be written to the data sets that are defined by the COPY1 and COPY2 DD statements.

```
//STEP1    EXEC DSNUPROC,UID='IUJMU107.MERGE1',
//          UTPROC='',SYSTEM='DSN'
//COPY1  DD DSN=IUJMU107.MERGE1.STEP1.COPY1,DISP=(MOD,CATLG,CATLG),
//          UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//COPY2  DD DSN=IUJMU107.MERGE1.STEP1.COPY2,DISP=(MOD,CATLG,CATLG),
//          UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSUT1   DD DSN=IUJMU107.MERGE1.STEP1.SYSUT1,DISP=(MOD,DELETE,CATLG),
//          UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
MERGECOPY TABLESPACE DSN8D81P.DSN8S81C
  COPYDDN      (COPY1,COPY2)
  NEWCOPY  NO
```

Figure 54. Example of creating a merged incremental copy

Example 2: Creating merged incremental copies and using templates. Each MERGECOPY control statement in Figure 55 specifies that MERGECOPY is to merge incremental image copies from the specified table space into a single incremental image copy for that table space. For each control statement, the COPYDDN option specifies that the output image copies are to be written to data sets that are defined by the T1 template. This template is defined in the TEMPLATE utility control statement. For more information about TEMPLATE utility control statements, see "Syntax and options of the TEMPLATE control statement " on page 593 in the TEMPLATE chapter.

```
//STEP1    EXEC DSNUPROC,UID='JULTU224.MERGE',
//         UTPROC='',
//         SYSTEM='SSTR'
//SYSUT1   DD DSN=JULTU224.MERGE.STEP1.SYSUT1,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
  TEMPLATE T1 UNIT(SYSDA)
              DSN(JULTU224.&SN..COPY&IC.&LOCREM.&PB..&JO.)
              DISP(NEW,CATLG,DELETE)
  MERGECOPY TABLESPACE DBLT2401.TPLT2401 DSNUM ALL NEWCOPY NO
          COPYDDN(T1)
  MERGECOPY TABLESPACE DBLT2401.TLLT24A1 DSNUM ALL NEWCOPY NO
          COPYDDN(T1)
  MERGECOPY TABLESPACE DBLT2401.TLLT24A2 DSNUM ALL NEWCOPY NO
          COPYDDN(T1)
  MERGECOPY TABLESPACE DBLT2401.TLLT24A3 DSNUM ALL NEWCOPY NO
          COPYDDN(T1)
  MERGECOPY TABLESPACE DBLT2401.TLLT24A4 DSNUM ALL NEWCOPY NO
          COPYDDN(T1)
```

Figure 55. Example of using templates

Example 3: Creating a merged full image copy. The control statement in Figure 56 on page 298 specifies that MERGECOPY is to merge all incremental image copies with the full image copy from table space DSN8S81C to create a new full image copy.

```
//STEP1    EXEC DSNUPROC,UID='IUJMU107.MERGE2',
//          UTPROC='',SYSTEM='DSN'
//COPY1  DD DSN=IUJMU107.MERGE2.STEP1.COPY1,DISP=(MOD,CATLG,CATLG),
//          UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//COPY2  DD DSN=IUJMU107.MERGE2.STEP1.COPY2,DISP=(MOD,CATLG,CATLG),
//          UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSUT1   DD DSN=IUJMU107.MERGE2.STEP1.SYSUT1,DISP=(MOD,DELETE,CATLG),
//          UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
MERGECOPY TABLESPACE DSN8D81P.DSN8S81C
  COPYDDN      (COPY1,COPY2)
  NEWCOPY  YES
```

Figure 56. Example of creating a merged full image copy

Chapter 18. MODIFY RECOVERY

The MODIFY utility with the RECOVERY option deletes records from the SYSIBM.SYSCOPY catalog table, related log records from the SYSIBM.SYSLGRNX directory table, and entries from the DBD, and recycles version numbers for reuse. You can remove records that were written before a specific date, or you can remove records of a specific age. You can delete records for an entire table space, partition, or data set.

You should run MODIFY regularly to remove outdated information from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX. These tables, and particularly SYSIBM.SYSLGRNX, can become very large and take up considerable amounts of space. By deleting outdated information from these tables, you can help improve performance for processes that access data from these tables.

The MODIFY RECOVERY utility automatically removes the SYSCOPY and SYSLGRNX recovery records that meet the age and date criteria for all indexes on the table space that were defined with the COPY YES attribute.

For a diagram of MODIFY RECOVERY syntax and a description of available options, see "Syntax and options of the MODIFY RECOVERY control statement" on page 300. For detailed guidance on running this utility, see "Instructions for running MODIFY RECOVERY" on page 302.

Output: MODIFY RECOVERY deletes image copy rows from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX.

For each full and incremental SYSCOPY record that is deleted from SYSIBM.SYSCOPY, the utility returns a message identifying the name of the copy data set.

For information about deleting SYSLGRNX rows, see "Deleting SYSLGRNX and SYSCOPY rows for a single partition or the entire table space" on page 303.

If MODIFY RECOVERY deletes at least one SYSCOPY record and the target table space or partition is not recoverable, the target object is placed in COPY-pending status.

For table spaces and indexes that are defined with COPY YES, the MODIFY RECOVERY utility updates the OLDEST_VERSION column of the following catalog tables:

- SYSIBM.SYSTABLESPACE
- SYSIBM.SYSTABLEPART
- SYSIBM.SYSINDEXES
- SYSIBM.SYSINDEXPART

For more information about how and when MODIFY RECOVERY updates these tables, see "The effect of MODIFY RECOVERY on version numbers" on page 305.

Authorization required: To execute this utility, you must use a privilege set that includes of the following authorities:

- IMAGCOPY privilege for the database to run MODIFY RECOVERY
- DBADM, DBCTRL, or DBMAINT authority for the database

- SYSCTRL or SYSADM authority

An ID with installation SYSOPR authority can also execute MODIFY RECOVERY, but only on a table space in the DSNDB01 or DSNDB06 database.

SYSIBM.SYSCOPY or SYSIBM.SYSLGRNX does not contain records for DSNDB06.SYSCOPY, DSNDB01.SYSUTILX, or DSNDB01.DBD01. You can run MODIFY RECOVERY on these table spaces, but you receive message DSNU573I, indicating that no SYSCOPY records could be found. No SYSCOPY or SYSLGRNX records are deleted.

Execution phases of MODIFY RECOVERY: The MODIFY RECOVERY phase operates in these phases:

UTILINIT Performs initialization and setup

MODIFY Deletes records

UTILTERM Performs cleanup

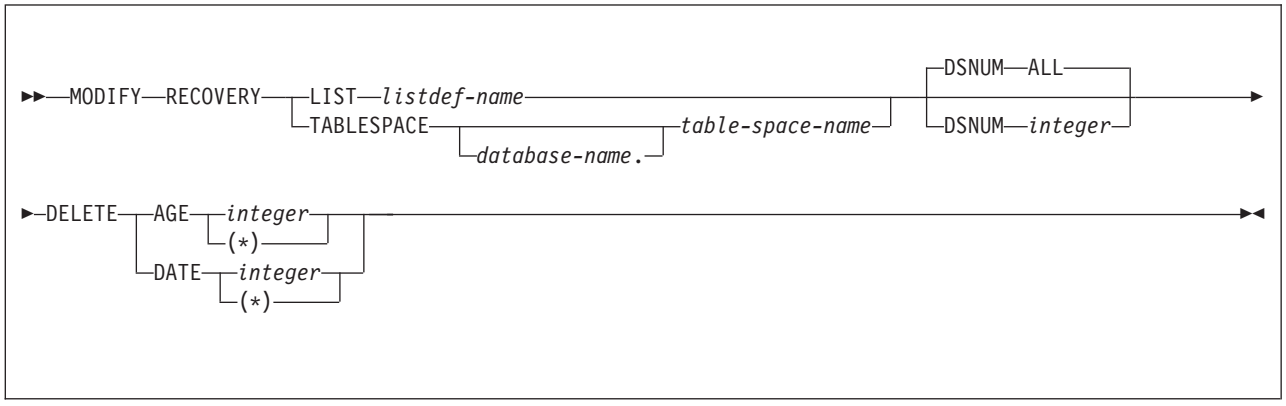
The following topics provide additional information:

- "Syntax and options of the MODIFY RECOVERY control statement"
- "Instructions for running MODIFY RECOVERY" on page 302
- "Concurrency and compatibility for MODIFY RECOVERY" on page 305
- "The effect of MODIFY RECOVERY on version numbers" on page 305
- "Sample MODIFY RECOVERY control statements" on page 306

Syntax and options of the MODIFY RECOVERY control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

LIST *listdef-name*
Specifies the name of a previously defined LISTDEF list name that contains only table spaces. You can specify one LIST keyword per MODIFY RECOVERY control statement. Do not specify LIST with the TABLESPACE keyword. MODIFY is invoked once for each table space in the list. For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

TABLESPACE *database-name.table-space-name*
Specifies the database and the table space for which records are to be deleted.

database-name Specifies the name of the database to which the table space belongs. *database-name* is optional. The **default** is **DSNDB04**.

table-space-name
Specifies the name of the table space.

DSNUM *integer*
Identifies a single partition or data set of the table space for which records are to be deleted; ALL deletes records for the entire data set and table space.

integer is the number of a partition or data set.

The **default** is **ALL**.

For a partitioned table space, *integer* is its partition number. The maximum is 4096.

For a nonpartitioned table space, use the data set integer at the end of the data set name as cataloged in the VSAM catalog. If image copies are taken by partition or data set and you specify DSNUM ALL, the table space is placed in COPY-pending status if a full image copy of the entire table space does not exist. The data set name has the following format, where *y* is either I or J, and *nnn* is the data set integer.

catname.DSNDB*x*.*dbname*.*tsname*.*y*0001.A*nnn*

If you specify DSNUM, MODIFY RECOVERY does not delete any SYSCOPY records for the partitions that have an RBA greater than that of the earliest point to which the entire table space could be recovered. That point might indicate a full image copy, a LOAD operation with LOG YES or a REORG operation with LOG YES.

If you specify DSNUM for a partitioned table space, MODIFY RECOVERY deletes SYSCOPY records for all partitioned index spaces as well as for the partition and updates the version numbers in the SYSIBM.SYSINDEXES catalog table. However, DB2 does not perform these functions for the nonpartitioned indexes.

See "Deleting SYSLGRNX and SYSCOPY rows for a single partition or the entire table space" on page 303 for more information about specifying DSNUM.

DELETE
Indicates that records are to be deleted. See the DSNUM description for restrictions on deleting partition statistics.

AGE *integer*
Deletes all SYSCOPY records that are older than a specified number of days.

integer is the number of days, and can range from 0 to 32767. Records that are created today are of age 0 and cannot be deleted by this option.

(*) deletes all records, regardless of their age.

DATE *integer*
Deletes all records that are written before a specified date.

integer can be in eight- or six-character format. You must specify a year (*yyyy* or *yy*), month (*mm*), and day (*dd*) in the form *yyyymmdd* or *yymmdd*. DB2 checks the system clock and converts six-character dates to the most recent, previous eight-character equivalent.

(*) deletes all records, regardless of the date on which they were written.

Instructions for running MODIFY RECOVERY

To run MODIFY RECOVERY you must:

1. Read "Before running MODIFY RECOVERY" in this chapter.
2. Prepare the necessary data sets, as described in "Data sets that MODIFY RECOVERY uses."
3. Create JCL statements, by using one of the methods described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for MODIFY RECOVERY, see "Sample MODIFY RECOVERY control statements" on page 306.)
4. Prepare a utility control statement that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks" on page 303.
5. Check the compatibility table in "Concurrency and compatibility for MODIFY RECOVERY" on page 305 if you want to run other jobs concurrently on the same target objects.
6. You can restart a MODIFY RECOVERY utility job, but it starts from the beginning again. You can terminate MODIFY RECOVERY with the TERM UTILITY command. For guidance in restarting online utilities, see "Terminating or restarting MODIFY RECOVERY" on page 304.
7. Run MODIFY RECOVERY by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Before running MODIFY RECOVERY

Recommendations for printing SYSCOPY records with REPORT RECOVERY: If you use MODIFY RECOVERY to delete SYSCOPY records, use the REPORT utility to view all SYSCOPY records for the object at the specified site to avoid deleting the wrong records.

Removing RECOVER-pending status: You cannot run MODIFY RECOVERY on a table space that is in RECOVER-pending status. See Chapter 23, "RECOVER," on page 355 for information about resetting the RECOVER-pending status.

Improving performance of MODIFY RECOVERY:To improve performance of
MODIFY RECOVERY and reduce contention on SYSLGRNX, it is recommended
that you run the REORG TABLESPACE utility on DSNDB01.SYSLGRNX on a
regular basis.

Data sets that MODIFY RECOVERY uses

Table 48 on page 303 lists the data sets that MODIFY RECOVERY uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 48. Data sets that MODIFY RECOVERY uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes

The following object is named in the utility control statement and does not require a DD statement in the JCL:

Table space
Object for which records are to be deleted.

Creating the control statement

See "Syntax diagram" on page 300 for MODIFY RECOVERY syntax and option descriptions. See "Sample MODIFY RECOVERY control statements" on page 306 for examples of MODIFY RECOVERY usage.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in your utility control statement:

Deleting SYSLGRNX and SYSCOPY rows for a single partition or the entire table space

Use the DSNUM option to specify whether SYSLGRNX and SYSCOPY rows should be deleted for a single partition or the entire table space. The DSNUM value that you should use (ALL or *integer*) depends on the type of image copies that exist for the table space. Use the following guidelines to determine whether to use DSNUM ALL or DSNUM *integer*:

- If image copies exist at only the partition level, use DSNUM *integer*.
- If image copies exist at only the data set level for a nonpartitioned table space, use DSNUM ALL. If DSNUM *integer* is used, SYSLGRNX records are not deleted.
- If image copies exist at only the table space or index space level, use DSNUM ALL.
- If image copies exist at both the partition level and the table space or index space level, use DSNUM ALL. **Restriction:** In this case, if you use DSNUM *integer*, MODIFY RECOVERY does not delete any SYSCOPY or SYSLGRNX records that are newer than the oldest recoverable point at the table space or index space level.
- If image copies exist at both the data set level and the table space level for a nonpartitioned table space, use DSNUM ALL. **Restriction:** In this case, if you use DSNUM *integer*, MODIFY RECOVERY does not delete any SYSCOPY or SYSLGRNX records that are newer than the oldest recoverable point at the table space level.

The preceding guidelines pertain to all image copies, regardless of how they were created, including those copies that were created by COPY, COPYTOCOPY, LOAD, REORG or MERGECOPY.

Deleting all image copy entries

MODIFY RECOVERY allows you to delete all image copy entries for a table space or data set. In this case, MODIFY RECOVERY:

- Issues message DSNU572I.
- Sets the COPY-pending restrictive status.
- Issues return code 4.

Deleting recovery rows for indexes

When you perform MODIFY RECOVERY on a table space, utility processing deletes SYSCOPY and SYSLGRNX rows that meet the AGE and DATE criteria for related indexes that were defined with COPY YES. If no SYSCOPY rows exist for an index, the RBA used for deletion of the table space SYSLGRNX rows is also used for deletion of the index SYSLGRNX rows.

Reclaiming space in the DBD

To reclaim space in the DBD when you drop a table, use the following procedure:

1. Commit the drop.
2. Run the REORG utility.
3. Run the COPY utility to make a full image copy of the table space.
4. Run MODIFY RECOVERY with the DELETE option to delete all previous image copies.

Deleting SYSOBDS entries

MODIFY RECOVERY removes entries for an object from the SYSOBDS catalog table when the last image copy that contains version 0 data rows or keys is deleted.

Improving REORG performance after adding a column

After you add a column to a table space, the next REORG of the table space creates default values for the added column by converting all of the fields in each row to the external DB2 format during the UNLOAD phase and then converting them to the internal DB2 format during the RELOAD phase. Subsequently, each REORG job for the table space repeats this processing in the UNLOAD and RELOAD phases. Use the following procedure to avoid repeating the compression cycle with each REORG:

1. Run the REORG utility on the table space.
2. Run the COPY utility to make a full image copy of the table space.
3. Run MODIFY RECOVERY with the DELETE option to delete all previous image copies. MODIFY RECOVERY changes the status of the column that is added after using the ALTER command only if SYSCOPY rows need to be deleted.

Terminating or restarting MODIFY RECOVERY

You can use the TERM UTILITY command to terminate MODIFY RECOVERY in any phase without any integrity exposure.

You can restart a MODIFY RECOVERY utility job, but it starts from the beginning again. For guidance in restarting online utilities, see “Restarting an online utility” on page 43.

Concurrency and compatibility for MODIFY RECOVERY

DB2 treats individual data and index partitions as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible.

Table 49 shows the restrictive state that the utility sets on the target object.

Table 49. Claim classes of MODIFY RECOVERY operations.

Target	MODIFY RECOVERY
Table space or partition	UTRW

Legend:

UTRW - Utility restrictive state - Read-write access allowed.

MODIFY RECOVERY can run concurrently on the same target object with any utility **except the following utilities**:

- COPY TABLESPACE
- LOAD
- MERGECOPY
- MODIFY RECOVERY
- RECOVER TABLESPACE
- REORG TABLESPACE

The target object can be a table space or partition.

The effect of MODIFY RECOVERY on version numbers

DB2 stores the range of used version numbers in the OLDEST_VERSION and CURRENT_VERSION columns of one or more of the following catalog tables, depending on the object:

- SYSIBM.SYSTABLESPACE
- SYSIBM.SYSTABLEPART
- SYSIBM.SYSINDEXES
- SYSIBM.SYSINDEXPART

The OLDEST_VERSION column contains the oldest used version number, and the CURRENT_VERSION column contains the current version number.

When you run MODIFY RECOVERY, the utility updates this range of used version numbers for table spaces and for indexes that are defined with the COPY YES attribute. MODIFY RECOVERY updates the OLDEST_VERSION column of the appropriate catalog table or tables with the version number of the oldest version that has not yet been applied to the entire object. DB2 can reuse any version numbers that are not in the range that is set by the values in the OLDEST_VERSION and CURRENT_VERSION columns.

Recycling of version numbers is required when all of the version numbers are being used. All version numbers are being used when one of the following situations is true:

- The value in the CURRENT_VERSION column is one less than the value in the OLDEST_VERSION column.

- The value in the CURRENT_VERSION column is 255 for table spaces or 15 for indexes, and the value in the OLDEST_VERSION column is 0 or 1.

To recycle version numbers for indexes that are defined with the COPY NO attribute, run LOAD REPLACE, REBUILD INDEX, REORG INDEX, or REORG TABLESPACE.

For more information about versions and how they are used by DB2, see Part 2 of *DB2 Administration Guide*.

Sample MODIFY RECOVERY control statements

Example 1: Deleting SYSCOPY records that are over a certain age. The following control statement specifies that the MODIFY RECOVERY utility is to delete all SYSCOPY records that are older than 90 days for table space DSN8D81A.DSN8S81E.

```
//STEP1    EXEC DSNUPROC,UID='IUIQU2UD.MODRCV1',
//            UTPROC='',SYSTEM='DSN'
//SYSIN    DD *
MODIFY RECOVERY TABLESPACE DSN8D81A.DSN8S81E DELETE AGE(90)
/*
```

Example 2: Deleting SYSCOPY records that are older than a certain date. The following control statement specifies that MODIFY RECOVERY is to delete all SYSCOPY records that were written before 10 September 2002.

```
MODIFY RECOVERY TABLESPACE DSN8D81A.DSN8S81D DELETE DATE(20020910)
```

Example 3: Deleting SYSCOPY records for partitions. The following control statements specifies that MODIFY RECOVERY is to delete the following SYSCOPY records for table space TU5AP053:

- Any records in partition 2 that are older than 5 days
- Any records in partition 3 that were written before 9 October 2001

```
//STEP2    EXEC  DSNUPROC,UID='FUN5U053.STEP2',UTPROC='',SYSTEM='SSTR'
//SYSIN DD *

    MODIFY RECOVERY TABLESPACE TU5AP053
           DSNUM 2
           DELETE AGE(5)

    MODIFY RECOVERY TABLESPACE TU5AP053
           DSNUM 3
           DELETE DATE(011009)
/*
```

Figure 57. Example MODIFY RECOVERY statements that delete SYSCOPY records for partitions

Example 4: Deleting all SYSCOPY records for objects in a list and viewing the results. In the following example job, the LISTDEF utility control statements define three lists (L1, L2, L3). The first group of REPORT utility control statements then specify that the utility is to report recovery information for the objects in these lists. Next, the MODIFY RECOVERY control statement specifies that the utility is to delete all SYSCOPY records for the objects in the L1 list. Finally, the second group of REPORT control statements specify that the utility is to report the recovery information for the same three lists. In this second report, no information will be reported for the objects in the L1 list because all of the SYSCOPY records

have been deleted.

```
//STEP4    EXEC DSNUPROC,UID='JULTU224.RCV1',
//            UTPROC='',SYSTEM='SSTR'
//SYSIN    DD *
   LISTDEF L1 INCLUDE TABLESPACE DBLT2401.T*
   LISTDEF L2 INCLUDE INDEXSPACE DBLT2401.I*
     LISTDEF  L3  INCLUDE INDEX IXLT2402
   REPORT RECOVERY TABLESPACE LIST L1
   REPORT RECOVERY INDEXSPACE LIST L2
     REPORT RECOVERY  INDEX LIST L3
   MODIFY RECOVERY LIST L1
        DELETE DATE(*)

   REPORT RECOVERY TABLESPACE LIST L1
   REPORT RECOVERY INDEXSPACE LIST L2
   REPORT RECOVERY INDEX LIST L3
/*
```

Figure 58. Example MODIFY RECOVERY statement that deletes all SYSCOPY records

For more information about the LISTDEF utility control statements, see Chapter 15, “LISTDEF,” on page 173. For more information about the REPORT utility control statements, see Chapter 27, “REPORT,” on page 525.

Chapter 19. MODIFY STATISTICS

The online MODIFY STATISTICS utility deletes unwanted statistics history records from the corresponding catalog tables. You can remove statistics history records that were written before a specific date, or you can remove records of a specific age. You can delete records for an entire table space, index space, or index.

Run MODIFY STATISTICS regularly to clear outdated information from the statistics history catalog tables. By deleting outdated information from those tables, you can improve performance for processes that access data from those tables.

For a diagram of MODIFY STATISTICS syntax and a description of available options, see "Syntax and options of the MODIFY STATISTICS control statement" on page 310. For detailed guidance on running this utility, see "Instructions for running MODIFY STATISTICS" on page 312.

Output: MODIFY STATISTICS deletes rows from the following catalog tables:
- SYSIBM.SYSCOLDIST_HIST
- SYSIBM.SYSCOLUMNS_HIST
- SYSIBM.SYSINDEXES_HIST
- SYSIBM.SYSINDEXPART_HIST
- SYSIBM.SYSINDEXSTATS_HIST
- SYSIBM.SYSLOBSTATS_HIST
- SYSIBM.SYSTABLEPART_HIST
- SYSIBM.SYSTABSTATS_HIST
- SYSIBM.SYSTABLES_HIST

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:
- STATS privilege for the database to run MODIFY STATISTICS
- DBADM, DBCTRL, or DBMAINT authority for the database
- SYSCTRL or SYSADM authority.

A user ID with installation SYSOPR authority can also execute MODIFY STATISTICS, but only on a table space in the DSNDB01 or DSNDB06 database.

Execution phases of MODIFY STATISTICS: The MODIFY STATISTICS utility operates in these phases:

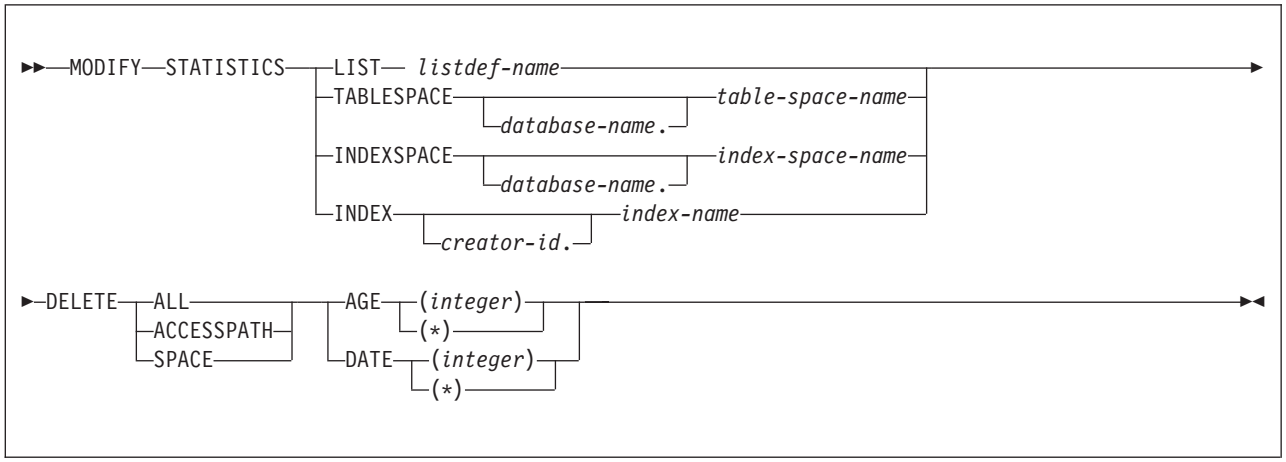
Phase	Description
UTILINIT	Performs initialization and setup
MODIFYS	Deletes records
UTILTERM	Performs cleanup

The following topics provide additional information:
- "Syntax and options of the MODIFY STATISTICS control statement" on page 310
- "Instructions for running MODIFY STATISTICS" on page 312
- "Concurrency and compatibility for MODIFY STATISTICS" on page 313
- "Sample MODIFY STATISTICS control statements" on page 314

Syntax and options of the MODIFY STATISTICS control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

LIST *listdef-name*
Specifies the name of a previously defined LISTDEF list name. You cannot repeat the LIST keyword or specify it with TABLESPACE, INDEXSPACE, or INDEX.

The list can contain index spaces, table spaces, or both. MODIFY STATISTICS is invoked once for each object in the list.

TABLESPACE *database-name.table-space-name*
Specifies the database and the table space for which catalog history records are to be deleted.

database-name Specifies the name of the database to which the table space belongs. *database-name* is optional. The **default** is **DSNDB04**.

table-space-name
Specifies the name of the table space for which statistics are to be deleted.

INDEXSPACE *database-name.index-space-name*
Specifies the qualified name of the index space for which catalog history information is to be deleted. The utility lists the name in the SYSIBM.SYSINDEXES table.

database-name Optionally specifies the name of the database to which the index space belongs. The **default** is **DSNDB04**.

index-space-name
Specifies the name of the index space for which the statistics are to be deleted.

INDEX *creator-id.index-name*
Specifies the index for which catalog history information is to be deleted.

creator-id
Optionally specifies the creator of the index. The **default** is **DSNDB04**.

index-name
Specifies the name of the index for which the statistics are to be deleted. Enclose the index name in quotation marks if the name contains a blank.

DELETE
Indicates that records are to be deleted.

ALL Deletes all statistics history rows that are related to the specified object from all catalog history tables.

Rows from the following history tables are deleted only when you specify DELETE ALL:

- SYSTABLES_HIST
- SYSTABSTATS_HIST
- SYSINDEXES_HIST
- SYSINDEXSTATS_HIST

ACCESSPATH
Deletes all access-path statistics history rows that are related to the specified object from the following history tables:
- SYSIBM.SYSCOLDIST_HIST
- SYSIBM.SYSCOLUMNS_HIST

SPACE
Deletes all space-tuning statistics history rows that are related to the specified object from the following history tables:
- SYSIBM.SYSINDEXPART_HIST
- SYSIBM.SYSTABLEPART_HIST
- SYSIBM.SYSLOBSTATS_HIST

AGE (*integer***)**
Deletes all statistics history rows that are related to the specified object and that are older than a specified number of days.

(*integer***)**
Specifies the number of days in a range from 0 to 32 767. This option cannot delete records that are created today (age 0).

(*) Deletes all records, regardless of their age.

DATE (*integer***)**
Deletes all records that are written before a specified date.

(*integer***)**
Specifies the date in an eight-character format. Specify a year (*yyyy*), month (*mm*), and day (*dd*) in the form *yyyymmdd*.

(*)
Deletes all records, regardless of the date on which they were written.

Instructions for running MODIFY STATISTICS

To run MODIFY STATISTICS you must:

1. Prepare the necessary data sets, as described in "Data sets that MODIFY STATISTICS uses."
2. Create JCL statements by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for MODIFY STATISTICS, see "Sample MODIFY STATISTICS control statements" on page 314.)
3. Prepare a utility control statement that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks."
4. Check the compatibility table in "Concurrency and compatibility for MODIFY STATISTICS" on page 313 if you want to run other jobs concurrently on the same target objects.
5. Restart a MODIFY STATISTICS utility job (it starts from the beginning again) or terminate MODIFY STATISTICS by using the TERM UTILITY command. For guidance in restarting online utilities, see "Restarting an online utility" on page 43.
6. Run MODIFY STATISTICS by using one of the methods described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Data sets that MODIFY STATISTICS uses

Table 50 lists the data sets that MODIFY STATISTICS uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 50. Data sets that MODIFY STATISTICS uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement	Yes
SYSPRINT	Output data set for messages	Yes

The following object is named in the utility control statement and does not require a DD statement in the JCL:

Table space or index space
Object for which records are to be deleted.

Creating the control statement

Create the utility control statement for the MODIFY STATISTICS job. See "Syntax diagram" on page 310 for MODIFY STATISTICS syntax and option descriptions. See "Sample MODIFY STATISTICS control statements" on page 314 for examples of MODIFY STATISTICS usage.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in your utility control statement.

Deciding which statistics history rows to delete

After analyzing trends by using the relevant historical catalog information and possibly taking actions based on this information, consider deleting all or part of

the statistics history catalog rows. Deleting outdated information from the statistics history catalog tables can improve performance for processes that access data from those tables. You also make available the space in the catalog. Then, the next time you update the relevant statistics by using RUNSTATS TABLESPACE, REBUILD INDEX, or REORG INDEX, DB2 repopulates the statistics history catalog tables with more recent historical data. Examining this data lets you determine the efficacy of any adjustments that you made as a result of your previous analysis.

Be aware that when you manually insert, update, or delete catalog information, DB2 does not store the historical information for those operations in the historical catalog tables.

Deleting specific statistics history rows

MODIFY STATISTICS lets you delete some or all statistics history rows for a table space, an index space, or an index.

You can choose to delete only the statistics rows that relate to access path selection by specifying the ACCESSPATH option. Alternatively, you can delete the rows that relate to space statistics by using the SPACE option. To delete rows in all statistics history catalog tables, including the SYSIBM.SYSTABLES_HIST catalog table, you must specify the DELETE ALL option in the utility control statement.

To delete statistics from the RUNSTATS history tables, you can either use the MODIFY STATISTICS utility or issue SQL DELETE statements. The MODIFY STATISTICS utility simplifies the purging of old statistics without requiring you to write the SQL DELETE statements. You can also delete rows that meet the age and date criteria by specifying the corresponding keywords (AGE and DATE) for a particular object.

To avoid time outs when you delete historical statistics with MODIFY STATISTICS, you should increase the LOCKMAX parameter for DSNDB06.SYSHIST with ALTER TABLESPACE.

Terminating or restarting MODIFY STATISTICS

You can use the TERM UTILITY command to terminate the MODIFY STATISTICS utility in any phase.

You can restart a MODIFY STATISTICS utility job, but it starts from the beginning again. For guidance in restarting online utilities, see “Restarting an online utility” on page 43.

Concurrency and compatibility for MODIFY STATISTICS

DB2 treats individual data and index partitions as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible.

Table 51 shows the restrictive state that the utility sets on the target object.

Table 51. Claim classes of MODIFY STATISTICS operations.

Target	MODIFY STATISTICS
Table space, index, or index space	UTRW

Table 51. Claim classes of MODIFY STATISTICS operations. (continued)

Target	MODIFY STATISTICS
Legend:	
UTRW - Utility restrictive state - read-write access allowed.	

Sample MODIFY STATISTICS control statements

Example 1: Deleting SYSIBM.SYSTABLES_HIST records by age. The following control statement specifies that the MODIFY STATISTICS utility is delete all statistics history records for table space DSN8D81A.DSN8S81E that are older than 60 days.

```
//STEP1    EXEC DSNUPROC,UID='IUIQU2UD.MODSTAT1',
//            UTPROC='',SYSTEM='DSN'
//SYSIN    DD *
  MODIFY STATISTICS TABLESPACE DSN8D81A.DSN8S81E
    DELETE ALL
    AGE 60
/*
```

Example 2: Deleting access path records for all objects in a list. The MODIFY STATISTICS control statement in Figure 59 specifies that the utility is to delete access-path statistics history rows that were created before 17 April 2000 for objects in the specified list. The list, M1, is defined in the preceding LISTDEF control statement and includes table spaces DB0E1501.TL0E1501 and DSN8D81A.DSN8S81E. For more information about LISTDEF control statements, see Chapter 15, “LISTDEF,” on page 173.

```
//STEP9    EXEC DSNUPROC,UID='JUOEU115.MDFYL9',
//         UTPROC='',
//         SYSTEM='SSTR'
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *

   LISTDEF M1 INCLUDE TABLESPACE DBOE1501.TLOE1501
              INCLUDE TABLESPACE DSN8D81A.DSN8S81E
   MODIFY STATISTICS LIST M1
                     DELETE ACCESSPATH DATE(20000417)
/*
```

Figure 59. MODIFY STATISTICS control statement that specifies that access path history records are to be deleted

Example 3: Deleting space-tuning statistics records for an index by age. The control statement in Figure 60 on page 315 specifies that MODIFY STATISTICS is to delete space-tuning statistics records for index ADMF001.IXOE15S1 that are older than one day.

```
//STEP9    EXEC DSNUPROC,UID='JUOEU115.MOFYS9',
//         UTPROC='',
//         SYSTEM='SSTR'
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *

   MODIFY STATISTICS INDEX ADMF001.IXOE15S1
                     DELETE SPACE AGE 1
/*
```

Figure 60. MODIFY STATISTICS control statement that specifies that space-tuning statistics records are to be deleted

Example 4: Deleting all statistics history records for an index space. The control statement in Figure 61 specifies that MODIFY STATISTICS is to delete all statistics history records for index space DBOE1501.IUOE1501. Note that the deleted records are not limited by date because (*) is specified.

```
//STEP8    EXEC DSNUPROC,UID='JUOEU115.MDFYL8',
//         UTPROC='',
//         SYSTEM='SSTR'
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
   MODIFY STATISTICS INDEXSPACE DBOE1501.IUOE1501
                     DELETE ALL DATE (*)
/*
```

Figure 61. MODIFY STATISTICS control statement that specifies that all statistics history records are to be deleted

Chapter 20. OPTIONS

The online OPTIONS utility control statement specifies processing options that are applicable across many utility executions in a job step. By specifying various options, you can:

- Preview utility control statements
- Preview LISTDEF or TEMPLATE definitions
- Override library names for LISTDEF lists or TEMPLATE definitions
- Specify how to handle errors during list processing
- Alter the return code for warning messages
- Restore all default options

See "Syntax and options of the OPTIONS control statement" for details.

Output: The OPTIONS control statement sets the specified processing options for the duration of the job step, or until replaced by another OPTIONS control statement within the same job step.

Authorization required: The OPTIONS control statement performs setup for subsequent control statements. The OPTIONS statement itself requires no privileges to execute.

Execution phases of OPTIONS: The OPTIONS control statement executes entirely in the UTILINIT phase, in which it performs setup for the subsequent utility.

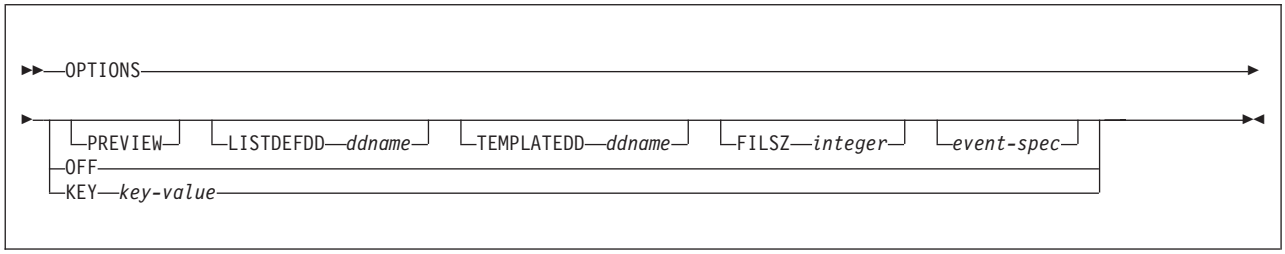
The following topics provide additional information:

- "Syntax and options of the OPTIONS control statement"
- "Instructions for using OPTIONS" on page 320
- "Concurrency and compatibility for OPTIONS" on page 321
- "Sample OPTIONS control statements" on page 321

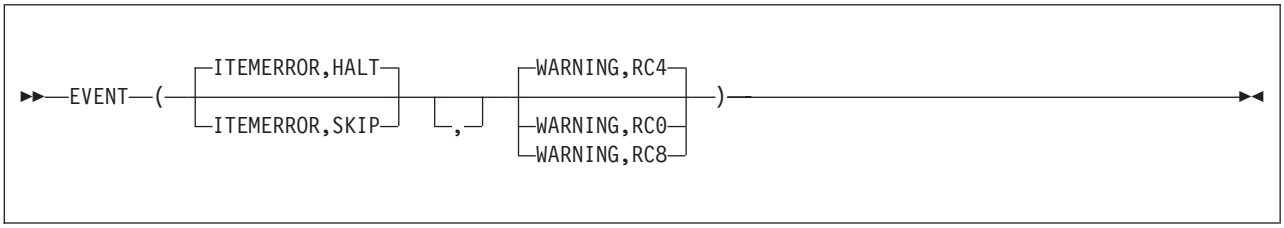
Syntax and options of the OPTIONS control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram



event-spec:



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

PREVIEW Specifies that the utility control statements that follow are to run in PREVIEW mode. The utility checks for syntax errors in all utility control statements, but normal utility execution does not take place. If the syntax is valid, the utility expands all LISTDEF lists and TEMPLATE DSNs that appear in SYSIN and prints results to the SYSPRINT data set.

PREVIEW evaluates and expands all LISTDEF statements into an actual list of table spaces or index spaces. It evaluates TEMPLATE DSNs and uses variable substitution for actual data set names when possible. It also expands lists from the SYSLISTD DD and TEMPLATE DSNs from the SYSTEMPL DD that a utility invocation references.

A definitive preview of TEMPLATE DSN values is not always possible. Substitution values for some variables, such as &DATE., &TIME., &SEQ. and &PART., can change at execution time. In some cases, PREVIEW generates approximate data set names. The OPTIONS utility substitutes unknown character variables with the character string ″UNKNOWN″ and unknown integer variables with zeroes.

Instead of OPTIONS PREVIEW, you can use a JCL PARM to activate preview processing. Although the two functions are identical, use JCL PARM to preview an existing set of utility control statements. Use the OPTION PREVIEW control statement when you invoke DB2 utilities through a stored procedure.

The JCL PARM is specified as the third JCL PARM of DSNUTILB and on the UTPROC variable of DSNUPROC, as shown in the following JCL:

```
//STEP1    EXEC DSNUPROC,UID='JULTU106.RECOVE1',
//         UTPROC='PREVIEW',SYSTEM='SSTR'
```

The PARM value PREVIEW causes the utility control statements in that job step to be processed for preview only. The LISTDEF and TEMPLATE control statements are expanded, but the utility does not execute.

OPTIONS PREVIEW is identical to the PREVIEW JCL parameter, except that you can specify a subsequent OPTIONS statement to turn off the preview for OPTIONS PREVIEW. Absence of the PREVIEW keyword in the OPTION control statement turns off

preview processing, but it does not override the PREVIEW JCL parameter, which, if specified, remains in effect for the entire job step.

LISTDEFDD *ddname*
Specifies the *ddname* of the LISTDEF definition library. The **default** value is **SYSLISTD**. A LISTDEF library is a data set that contains only LISTDEF control statements. This data set is processed only when a referenced LIST is not found in SYSIN. See Chapter 15, "LISTDEF," on page 173 for details.

TEMPLATEDD *ddname*
Specifies the *ddname* of the TEMPLATE definition library. The **default** value is **SYSTEMPL**. A TEMPLATE library is a data set that contains only TEMPLATE control statements. This data set is processed only when a referenced name does not exist in the job step as a DD name and is not found in SYSIN as a TEMPLATE name. See Chapter 31, "TEMPLATE," on page 593 TEMPLATE CONTROL STATEMENT for details.

FILSZ *integer* Specifies a file size in megabytes and overrides the DFSORT file size. Only use this keyword under the direction of IBM Software Support.

EVENT Specifies one or more pairs of utility processing events and the matching action for the event. Not all actions are valid for all events.

The parentheses and commas in the EVENT operand are currently optional but they may be required in a future release.

ITEMERROR

Specifies how utility processing is to handle errors during list processing. Specifically, this keyword indicates the effect on processing in response to return code 8. By default, utility processing stops (HALT). The ITEMERROR event does not include abnormal terminations (abends).

Note that for the QUIESCE utility, the indexes for the table spaces in the list, if any, are considered as list items for the purposes of the ITEMERROR event. ITEMERROR affects how errors are handled on both the table spaces and the indexes.

HALT
Specifies that the utility is to stop after the event.

SKIP
Ignores the event and skips the list item. Processing continues with the next item in the list.

SKIP applies only during the processing of a valid list. SKIP does not apply if a utility detects that a list is not valid for the utility that is invoked. In that case, the list is rejected with an error message and the processing of the list is not initiated.

If any of the items in a list is skipped, the utility produces a return code of 8, which terminates the job step. The following code shows an OPTIONS statement with the SKIP option:

```
OPTIONS EVENT (ITEMERROR, SKIP)
COPY LISTA
COPY LISTB
```

If LISTA contains ten objects and one object produces a return code 8 during the COPY, the other nine objects in the list are copied successfully. The job step ends with a return code 8 and COPY LISTB is not executed.

WARNING Specifies a response to the return code message event.

Use WARNING to alter the return code for warning messages. You can alter the return code from message DSNU010I with this option. If you alter the message return code, message DSNU1024I is issued to document the new return code.

Action choices are as follows:

RC0
Lowers the final return code of a single utility invocation that ends in a return code 4 to a return code of 0. Use RC0 to force a return code of 0 for warning messages.

Use this option only when return code 4 is expected, is acceptable, and other mechanisms are in place to validate the results of a utility execution.

RC4
Specifies that return codes for warning messages are to remain unchanged. Use RC4 to override a previous OPTIONS WARNING specification in the same job step.

RC8
Raises the final return code of a single utility invocation that ends in a return code 4 to a return code of 8. Use RC8 to force a return code of 8 for warning messages. The return code of 8 causes the job step to terminate and subsequent utility control statements are not executed.

OFF Specifies that all default options are to be restored. OPTIONS OFF does not override the PREVIEW JCL parameter, which, if specified, remains in effect for the entire job step. You cannot specify any other OPTIONS keywords with OPTIONS OFF.

OPTIONS OFF is equivalent to OPTIONS LISTDEFDD SYSLISTD TEMPLATEDD SYSTEMPL EVENT (ITEMERROR, HALT, WARNING, RC4).

KEY Specifies an option that you should use only when you are instructed by IBM Software Support. OPTIONS KEY is followed by a single operand that IBM Software Support provides when needed.

Instructions for using OPTIONS

Executing statements in preview mode

To execute utility control statements in preview mode, use OPTIONS PREVIEW. Control statements are previewed for use with LISTDEF lists and TEMPLATE definitions but the specified options are not actually executed.

Specifying LISTDEF and TEMPLATE libraries

To use different LISTDEF and TEMPLATE libraries, specify OPTIONS LISTDEFDD and OPTIONS TEMPLATEDD to override the names of the optional library data sets.

Overriding standard utility processing behavior

To alter settings for warning return codes and error handling during list processing, use OPTIONS EVENT to override the standard utility processing behaviors.

Using Multiple OPTIONS control statements

You can repeat an OPTIONS control statement within the SYSIN DD statement. If you repeat the control statement, it entirely replaces any prior OPTIONS control statement.

Terminating or restarting OPTIONS

You can terminate an OPTIONS utility job by using the TERM UTILITY command if you submitted the job or have SYSOPR, SYSCTRL, or SYSADM authority.

You can restart an OPTIONS utility job, but it starts from the beginning again. If you are restarting this utility as part of a larger job in which OPTIONS completed successfully, but a later utility failed, do not change the OPTIONS utility control statement, if possible. If you must change the OPTIONS utility control statement, use caution; any changes can cause the restart processing to fail. For example, if you specify a valid OPTIONS statement in the initial invocation, and then on restart, specify OPTIONS PREVIEW, the job fails. For guidance in restarting online utilities, see “Restarting an online utility” on page 43.

Concurrency and compatibility for OPTIONS

OPTIONS is a utility control statement that you can use to set up an environment for another utility to follow. The OPTIONS statement is stored until a specific utility references the statement. When referenced by another utility, the list is expanded. At that time, the concurrency and compatibility restrictions of that utility apply, with the additional restriction that the catalog tables that are necessary to expand the list must be available for read-only access.

Sample OPTIONS control statements

Example 1: Checking control statement syntax and previewing lists and TEMPLATE data set names. The following OPTIONS utility control statement specifies that the subsequent utility control statements are to run in PREVIEW mode. In PREVIEW mode, DB2 checks for syntax errors in all utility control statements, but normal utility execution does not take place. If the syntax is valid, DB2 expands the CPYLIST list and the data set names in the COPYLOC and COPYREM TEMPLATE utility control statements and prints these results to the SYSPRINT data set.

```
OPTIONS PREVIEW
TEMPLATE COPYLOC UNIT(SYSDA)
         DSN(&DB..&TS..D&JDATE..&STEPNAME..COPY&IC.&LOCREM.&PB.)
         DISP(NEW,CATLG,CATLG) SPACE(200,20) TRK
         VOLUMES(SCR03)
TEMPLATE COPYREM UNIT(SYSDA)
         DSN(&DB..&TS..&UT..T&TIME..COPY&IC.&LOCREM.&PB.)
         DISP(NEW,CATLG,CATLG) SPACE(100,10) TRK
LISTDEF CPYLIST INCLUDE TABLESPACES DATABASE DBLT0701
COPY LIST CPYLIST FULL YES
     COPYDDN(COPYLOC,COPYLOC)
     RECOVERYDDN(COPYREM,COPYREM)
     SHRLEVEL REFERENCE
```

Figure 62. Example OPTIONS statement for checking syntax and previewing lists and templates.

Example 2: Specifying LISTDEF and TEMPLATE definition libraries. In the following example, the OPTIONS control statements specify the DD names of the LISTDEF definition libraries and the TEMPLATE definition libraries.

The first OPTIONS statement specifies that the LISTDEF definition library is identified by the V1LIST DD statement and the TEMPLATE definition library is identified by the V1TEMPL DD statement. These definition libraries apply to the subsequent COPY utility control statement. Therefore, if DB2 does not find the PAYTBSP list in SYSIN, it searches the V1LIST library, and if DB2 does not find the PAYTEMP1 template in SYSIN, it searches the V1TEMP library.

The second OPTIONS statement is similar to the first, but it identifies different libraries and applies to the second COPY control statement. This second COPY control statement looks similar to the first COPY job. However, this statement processes a different list and uses a different template. Whereas the first COPY job uses the PAYTBSP list from the V1LIST library, the second COPY job uses the PAYTBSP list from the V2LIST library. Also, the first COPY job uses the PAYTEMP1 template from the V1TEMPL library, the second COPY job uses the PAYTEMP1 template from the V2TEMPL library.

```
OPTIONS LISTDEFDD V1LIST TEMPLATEDD V1TEMPL
COPY LIST PAYTBSP COPYDDN(PAYTEMP1,PAYTEMP1)

OPTIONS LISTDEFDD V2LIST TEMPLATEDD V2TEMPL
COPY LIST PAYTBSP COPYDDN(PAYTEMP1,PAYTEMP1)
```

Example 3: Forcing a return code 0. In the following example, the first OPTIONS control statement forces a return code of 0 for the subsequent MODIFY RECOVERY utility control statement. Ordinarily, this statement ends with a return code of 4 because it specifies that DB2 is to delete all SYSCOPY records for table space A.B. The second OPTIONS control statement restores the default options, so that no return codes will be overridden for the second MODIFY RECOVERY control statement.

```
OPTIONS EVENT(WARNING,RC0)
MODIFY RECOVERY TABLESPACE A.B DELETE AGE(*)
OPTIONS OFF
MODIFY RECOVERY TABLESPACE C.D DELETE AGE(30)
```

Example 4: Checking syntax and skipping errors while processing list objects. In Figure 63 on page 323, the first OPTIONS utility control statement specifies that the subsequent utility control statements are to run in PREVIEW mode. In PREVIEW mode, DB2 checks for syntax errors in all utility control statements, but normal

utility execution does not take place. If the syntax is valid, DB2 expands the three lists (LIST1_LISTDEF, LIST2_LISTDEF, and LIST3_LISTDEF) and prints these results to the SYSPRINT data set.

The second OPTIONS control statement specifies how DB2 is to handle return codes of 8 in any subsequent utility statements that process a valid list. If processing of a list item produces return code 8, DB2 skips that item, and continues to process the rest of the items in the list, but DB2 does not process the next utility control statement. Instead, the job ends with return code 8.

```
OPTIONS PREVIEW
LISTDEF COPY1_LISTDEF
            INCLUDE TABLESPACES TABLESPACE DSNDB01.SPT01
            INCLUDE TABLESPACES TABLESPACE DSNDB06.SYSDBASE
            INCLUDE TABLESPACES TABLESPACE DSNDB06.SYSDBAUT
            INCLUDE TABLESPACES TABLESPACE DSNDB06.SYSGPAUT
            INCLUDE TABLESPACES TABLESPACE DSNDB06.SYSGROUP
            INCLUDE TABLESPACES TABLESPACE DBA91302.T?A9132*
LISTDEF COPY2_LISTDEF
            INCLUDE TABLESPACES TABLESPACE DBA91303.TLA9133A
            INCLUDE TABLESPACES TABLESPACE DBA91303.TSA9133B
            INCLUDE TABLESPACES TABLESPACE DBA91303.TPA9133C
            INCLUDE TABLESPACES TABLESPACE DBA91304.TLA9134A
            INCLUDE TABLESPACES TABLESPACE DSNDB06.SYSUSER
            INCLUDE TABLESPACES TABLESPACE DSNDB06.SYSVIEWS
            INCLUDE TABLESPACES TABLESPACE DSNDB06.SYSSTATS
            INCLUDE TABLESPACES TABLESPACE DSNDB06.SYSDDF
            INCLUDE TABLESPACES TABLESPACE DSNDB06.SYSOBJ
LISTDEF COPY3_LISTDEF
            INCLUDE TABLESPACES TABLESPACE DBA91304.TSA9134B
            INCLUDE TABLESPACES TABLESPACE DSNDB06.SYSHIST
            INCLUDE TABLESPACES TABLESPACE DSNDB06.SYSGRTNS
            INCLUDE TABLESPACES TABLESPACE DSNDB06.SYSJAVA
            INCLUDE TABLESPACES TABLESPACE DBA91304.TPA9134C
OPTIONS EVENT(ITEMERROR,SKIP)
TEMPLATE TMP1 UNIT(SYSDA) DISP(MOD,CATLG,CATLG)
         VOLUMES(SCR03)
         DSN(DH109013.&TS..COPY&ICTYPE.&LOCREM.&PRIBAC.)
COPY LIST COPY1_LISTDEF SHRLEVEL REFERENCE
     COPYDDN (TMP1)
     RECOVERYDDN (TMP1)
     FULL YES
COPY LIST COPY2_LISTDEF SHRLEVEL REFERENCE
     COPYDDN (TMP1,TMP1)
     FULL YES
COPY LIST COPY3_LISTDEF SHRLEVEL REFERENCE
     COPYDDN (TMP1,TMP1)
     RECOVERYDDN (TMP1,TMP1)
     FULL YES
```

Figure 63. Example OPTIONS statements for checking syntax and skipping errors

Chapter 21. QUIESCE

The online QUIESCE utility establishes a *quiesce point* for a table space, partition, table space set, or list of table spaces and table space sets. A quiesce point is the current log RBA or log record sequence number (LRSN). QUIESCE then records the quiesce point in the SYSIBM.SYSCOPY catalog table. A successful QUIESCE improves the probability of a successful RECOVER or COPY. You should run QUIESCE frequently between regular executions of COPY to establish regular recovery points for future point-in-time recovery.

For a diagram of QUIESCE syntax and a description of available options, see "Syntax and options of the QUIESCE control statement" on page 326. For detailed guidance on running this utility, see "Instructions for running QUIESCE" on page 327.

Output: With the WRITE(YES) option, QUIESCE writes changed pages for the table spaces and their indexes from the DB2 buffer pool to disk. The catalog table SYSCOPY records the current RBA and the timestamp of the quiesce point. A row with ICTYPE='Q' is inserted into SYSIBM.SYSCOPY for each table space that is quiesced. DB2 also inserts a SYSCOPY row with ICTYPE='Q' for any indexes (defined with the COPY YES attribute) over a table space that is being quiesced. (Table spaces DSNDB06.SYSCOPY, DSNDB01.DBD01, and DSNDB01.SYSUTILX are an exception; their information is written to the log.)

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:
- IMAGCOPY privilege for the database
- DBADM, DBCTRL, or DBMAINT authority for the database
- SYSCTRL or SYSADM authority

An ID with installation SYSOPR authority can also execute QUIESCE, but only on a table space in the DSNDB01 or DSNDB06 database.

You can specify DSNDB01.SYSUTILX, but you cannot include it in a list with other table spaces to be quiesced. Recover to current of the catalog/directory table spaces is preferred and recommended. However, if a point-in-time recovery of the catalog/directory table spaces is desired, a separate quiesce of DSNDB06.SYSCOPY is required after a quiesce of the other catalog/directory table spaces.

Execution phases of QUIESCE: The QUIESCE utility operates in these phases:

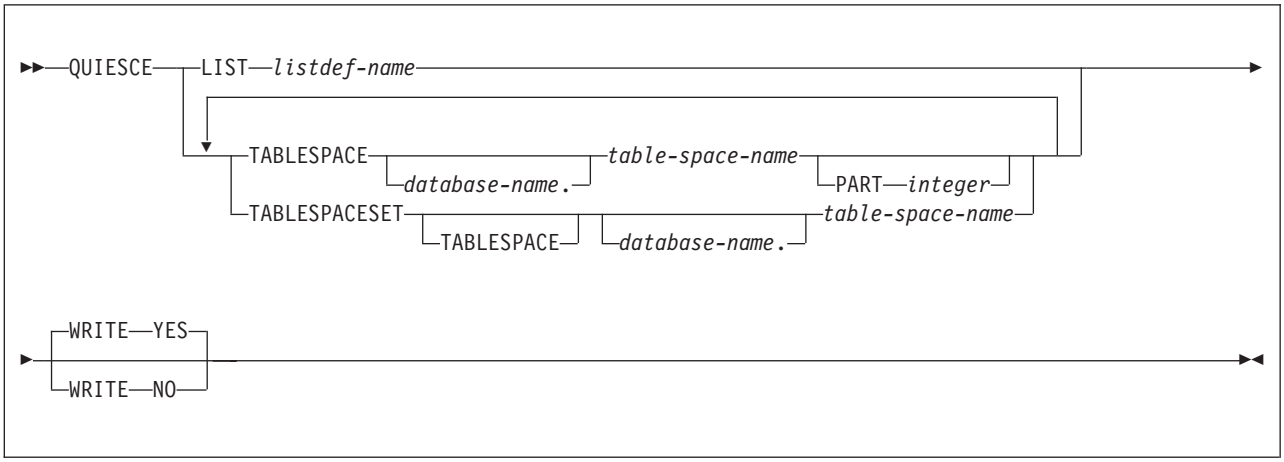
Phase	Description
UTILINIT	Initialization and setup
QUIESCE	Determining the quiesce point and updating the catalog
UTILTERM	Cleanup

The following topics provide additional information:
- "Syntax and options of the QUIESCE control statement" on page 326
- "Instructions for running QUIESCE" on page 327
- "Concurrency and compatibility for QUIESCE" on page 330
- "Sample QUIESCE control statements" on page 331

Syntax and options of the QUIESCE control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

LIST *listdef-name*

Specifies the name of a previously defined LISTDEF list name that contains only table spaces. The utility allows one LIST keyword for each QUIESCE control statement. Do not specify LIST with the TABLESPACE or TABLESPACESET keyword. QUIESCE is invoked once for the entire list. For the QUIESCE utility, the related index spaces are considered to be list items for the purposes of OPTIONS ITEMERROR processing. You can alter the utility behavior during processing of related indexes with the OPTIONS ITEMERROR statement. For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

TABLESPACE *database-name.table-space-name*

For QUIESCE TABLESPACE, specifies the table space that is to be quiesced.

For QUIESCE TABLESPACESET, specifies a table space in the table space set that is to be quiesced. For QUIESCE TABLESPACESET, the TABLESPACE keyword is optional.

database-name

Optionally specifies the name of the database to which the table space belongs. The **default** is **DSNDB04**.

table-space-name

Specifies the name of the table space that is to be quiesced. You can specify DSNDB01.SYSUTILX, but do not include that name in a list with other table spaces that are to be quiesced. If a point-in-time

recovery is planned for the catalog and directory, DSNDB06.SYSCOPY must be quiesced separately after all other catalog and directory table spaces.

PART *integer*
Identifies a partition that is to be quiesced.

integer is the number of the partition and must be in the range from 1 to the number of partitions that are defined for the table space. The maximum is 4096.

TABLESPACESET
Indicates that all of the referentially related table spaces in the table space set are to be quiesced. For the purposes of the QUIESCE utility, a table space set is one of these:
- A group of table spaces that have a referential relationship
- A base table space with all of its LOB table spaces

WRITE
Specifies whether the changed pages from the table spaces and index spaces are to be written to disk.

YES
Establishes a quiesce point and writes the changed pages from the table spaces and index spaces to disk. The **default** is **YES**.

NO
Establishes a quiesce point but does not write the changed pages from the table spaces and index spaces to disk.

Instructions for running QUIESCE

To run QUIESCE, you must:

1. Read "Before running QUIESCE" in this section.
2. Prepare the necessary data sets, as described in "Data sets that QUIESCE uses" on page 328.
3. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for QUIESCE, see "Sample QUIESCE control statements" on page 331.)
4. Prepare a utility control statement that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks" on page 328.
5. Check the compatibility table in "Concurrency and compatibility for QUIESCE" on page 330 if you want to run other jobs concurrently on the same target objects.
6. Plan for restart if the QUIESCE job doesn't complete, as described in "Terminating or restarting QUIESCE" on page 330.
7. Run QUIESCE by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Before running QUIESCE

You cannot run QUIESCE on a table space that is in COPY-pending, CHECK-pending, RECOVER-pending, or auxiliary CHECK-pending status. See "Resetting COPY-pending status" on page 269, "Resetting CHECK-pending status" on page 72, "Resetting REBUILD-pending status" on page 270, and Appendix C, "Advisory or restrictive states," on page 853 for information about resetting these statuses.

Data sets that QUIESCE uses

Table 52 lists the data sets that QUIESCE uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 52. Data sets that QUIESCE uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes

The following object is named in the utility control statement and does not require a DD statement in the JCL:

Table space
Object that is to be quiesced. (If you want to quiesce only one partition of a table space, you must use the PART option in the control statement.)

Creating the control statement

Create the utility control statement for the QUIESCE job. See "Syntax and options of the QUIESCE control statement" on page 326 for QUIESCE syntax and option descriptions. See "Sample QUIESCE control statements" on page 331 for examples of QUIESCE usage.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in your utility control statement:

Using QUIESCE for recovery

You can recover a table space to its quiesce point by using the RECOVER TABLESPACE utility. See Chapter 23, "RECOVER," on page 355 for information about the RECOVER TABLESPACE utility.

Using QUIESCE on catalog and directory objects

You can quiesce DSNDB01.SYSUTILX, but DSNDB01.SYSUTILX must be the only table space in the QUIESCE control statement.

If a point-in-time recovery is planned for the catalog and directory, a separate QUIESCE control statement for DSNDB06.SYSCOPY is required after you quiesce the other catalog and directory table spaces. A separate QUIESCE of DSNDB06.SYSCOPY is needed after the QUIESCE of other objects to ensure that a subsequent point-in-time recovery of DSNDB06.SYSCOPY recovers all of the QUIESCE SYSCOPY records for the other catalog and directory objects.

Obtaining a common quiesce point

Use the QUIESCE utility with the TABLESPACESET option to obtain a common quiesce point for related table spaces. For the purposes of the QUIESCE utility, a table space set is:

- A group of table spaces that have a referential relationship
- A base table space with all of its LOB table spaces

If you use QUIESCE TABLESPACE instead and do not include every member, you might encounter problems when you run RECOVER on the table spaces in the structure. RECOVER checks if a complete table space set is recovered to a single point in time. If the complete table space set is not recovered to a single point in time, RECOVER places all dependent table spaces into CHECK-pending status.

You should QUIESCE and RECOVER the LOB table spaces to the same point in time as the associated base table space. A group of table spaces that have a referential relationship should all be quiesced to the same point in time.

When you use QUIESCE WRITE YES on a table space, the utility inserts a SYSCOPY row that specifies ICTYPE='Q' for each related index that is defined with COPY=YES in order to record the quiesce point.

Specifying a list of table spaces and table space sets

You can specify as many objects in your QUIESCE job as can be allowed by available memory in the batch address space and in the DB2 DBM1 address space.

Be aware of the following considerations when you specify a list of objects to quiesce:

- Each table space set is expanded into a list of table spaces that have a referential relationship, or into a list that contains a base table space with all of its LOB table spaces.
- If you specify a list of table spaces or table space sets to quiesce and duplicate a table space, utility processing continues, and the table space is quiesced only once. QUIESCE issues return code 4 and warning message DSNU533I to alert you of the duplication.
- If you specify the same table space twice in a list, using PART *n* in one specification, and PART *m* for the other specification, each partition is quiesced once.

Running QUIESCE on a table space in pending status

If you run QUIESCE on a table space in COPY-pending, CHECK-pending, or RECOVER-pending status, it terminates with messages that are similar to those messages shown in Figure 64.

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = R92341Q
DSNU050I    DSNUGUTC - QUIESCE TABLESPACE UTQPD22A.UTQPS22D
                               TABLESPACE UTQPD22A.UTQPS22E
                               TABLESPACE UTQPD22A.EMPPROJA
DSNU471I % DSNUQUIA COPY PENDING ON TABLESPACE UTQPD22A.EMPPROJA PROHIBITS
           PROCESSING
DSNU012I DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

Figure 64. Termination messages when you run QUIESCE on a table space with pending restrictions

When you run QUIESCE on a table space or index space that is in COPY-pending, CHECK-pending, or RECOVER-pending status, you might also receive one or

more of the messages that are shown in Figure 65.

```
DSNU202I csect RECOVER PENDING ON TABLESPACE... PROHIBITS PROCESSING
DSNU203I csect  RECOVER PENDING ON INDEX ... PROHIBITS PROCESSING
DSNU204I csect  PAGESET REBUILD PENDING ON INDEX ... PROHIBITS PROCESSING
DSNU208I csect   GROUP BUFFER POOL RECOVER PENDING ON INDEX ... PROHIBITS PROCESSING
DSNU209I csect  RESTART PENDING ON ... PROHIBITS PROCESSING
DSNU210I csect  INFORMATIONAL COPY PENDING ON INDEX ... PROHIBITS PROCESSING
DSNU211I csect  CHECK PENDING ON  ... PROHIBITS PROCESSING
DSNU214I csect  REBUILD PENDING ON INDEX ... PROHIBITS PROCESSING
DSNU215I csect  REFRESH PENDING ON  ... PROHIBITS PROCESSING
DSNU471I csect  COPY PENDING ON TABLESPACE ... PROHIBITS PROCESSING
DSNU568I csect  INDEX ... IS IN INFORMATIONAL COPY PENDING
```

Figure 65. Messages for pending restrictions on QUIESCE

Determining why the write to disk fails

QUIESCE attempts to write pages of each table space to disk. If any of the following conditions is encountered, the write to disk fails:

- The table space has a write error range.
- The table space has deferred restart pending.
- An I/O error occurs.

If any of the preceding conditions is true, QUIESCE terminates with a return code of 4 and issues a DSNU473I warning message.

Terminating or restarting QUIESCE

If you use TERM UTILITY to terminate QUIESCE when it is active, QUIESCE releases the drain locks on table spaces. If QUIESCE is stopped, the drain locks have already been released.

You can restart a QUIESCE utility job, but it starts from the beginning again. For guidance in restarting online utilities, see “Restarting an online utility” on page 43.

Concurrency and compatibility for QUIESCE

DB2 treats individual data and index partitions as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible.

Table 53 shows which claim classes QUIESCE drains and any restrictive state that the utility sets on the target object.

Table 53. Claim classes of QUIESCE operations.

Target	WRITE YES	WRITE NO
Table space or partition	DW/UTRO	DW/UTRO
Partitioning index, data-partitioned secondary index, or partition	DW/UTRO	
Nonpartitioned secondary index	DW/UTRO	

Legend:
- DW - Drain the write claim class - concurrent access for SQL readers
- UTRO - Utility restrictive state - read-only access allowed

Table 54 on page 331 shows which utilities can run concurrently with QUIESCE on the same target object. The target object can be a table space, an index space, or a partition of a table space or index space. If compatibility depends on particular

options of a utility, that information is also documented in the table. QUIESCE does not set a utility restrictive state if the target object is DSNDB01.SYSUTILX.

Table 54. Compatibility of QUIESCE with other utilities

Action	Compatible with QUIESCE?
CHECK DATA DELETE NO	Yes
CHECK DATA DELETE YES	No
CHECK INDEX	Yes
CHECK LOB	Yes
COPY INDEXSPACE SHRLEVEL CHANGE	No
COPY INDEXSPACE SHRLEVEL REFERENCE	Yes
COPY TABLESPACE SHRLEVEL CHANGE	No
COPY TABLESPACE SHRLEVEL REFERENCE	Yes
DIAGNOSE	Yes
LOAD	No
MERGECOPY	Yes
MODIFY	Yes
QUIESCE	Yes
REBUILD INDEX	No
RECOVER INDEX	No
RECOVER TABLESPACE	No
REORG INDEX	No
REORG TABLESPACE UNLOAD CONTINUE or PAUSE	No
REORG TABLESPACE UNLOAD ONLY or EXTERNAL	Yes
REPAIR DELETE or REPLACE	No
REPAIR DUMP or VERIFY	Yes
REPORT	Yes
RUNSTATS	Yes
STOSPACE	Yes
UNLOAD	Yes

To run the QUIESCE utility on DSNDB01.SYSUTILX, ensure that QUIESCE is the only utility in the job step.

QUIESCE on SYSUTILX is an exclusive job; such a job can interrupt another job between job steps, possibly causing the interrupted job to time out.

Sample QUIESCE control statements

Example 1: Establishing a quiesce point for three table spaces. The following control statement specifies that the QUIESCE utility is to establish a quiesce point for table spaces DSN8D81A.DSN8S81D, DSN8D81A.DSN8S81E, and DSN8D81A.DSN8S81P.

```
//STEP1    EXEC DSNUPROC,UID='IUIQU2UD.QUIESC2',
//            UTPROC='',SYSTEM='DSN'
//SYSIN    DD *
```

```
QUIESCE TABLESPACE DSN8D81A.DSN8S81D
        TABLESPACE DSN8D81A.DSN8S81E
        TABLESPACE DSN8D81A.DSN8S81P
//*
```

Figure 66. shows the output that the preceding command produces.

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    DSNUGUTC -  QUIESCE TABLESPACE DSN8D81A.DSN8S81D
                            TABLESPACE DSN8D81A.DSN8S81E
                            TABLESPACE DSN8D81A.DSN8S81P
DSNU477I  = DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D81A.DSN8S81D
DSNU477I  = DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D81A.DSN8S81E
DSNU477I  = DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D81A.DSN8S81P
DSNU474I  = DSNUQUIA - QUIESCE AT RBA 000004E43B78 AND AT LRSN 000004E43B78
DSNU475I    DSNUQUIB - QUIESCE UTILITY COMPLETE, ELAPSED TIME= 00:00:02
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 66. Example output from a QUIESCE job that establishes a quiesce point for three table spaces

Example 2: Establishing a quiesce point for a list of objects. In the following example, the QUIESCE control statement uses a list to specify that the QUIESCE utility is to establish a quiesce point for the same table spaces as in example 1. The list is defined in the LISTDEF utility control statement.

```
//STEP1    EXEC DSNUPROC,UID='IUIQU2UD.QUIESC2',
//            UTPROC='',SYSTEM='DSN'
//SYSIN    DD *
//DSNUPROC.SYSIN    DD  *
LISTDEF QUIESCELIST INCLUDE TABLESPACE DSN8D81A.DSN8S81D
                    INCLUDE TABLESPACE DSN8D81A.DSN8S81E
                    INCLUDE TABLESPACE DSN8D81A.DSN8S81P
QUIESCE LIST QUIESCELIST
//*
```

Figure 67. shows the output that the preceding command produces.

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    DSNUGUTC -  LISTDEF QUIESCELIST INCLUDE TABLESPACE DSN8D81A.DSN8S81D
INCLUDE TABLESPACE DSN8D81A.DSN8S81E
INCLUDE TABLESPACE DSN8D81A.DSN8S81P
DSNU1035I   DSNUILDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    DSNUGUTC -  QUIESCE LIST QUIESCELIST
DSNU477I  = DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D81A.DSN8S81D
DSNU477I  = DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D81A.DSN8S81E
DSNU477I  = DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D81A.DSN8S81P
DSNU474I  = DSNUQUIA - QUIESCE AT RBA 000004E56419 AND AT LRSN 000004E56419
DSNU475I    DSNUQUIB - QUIESCE UTILITY COMPLETE, ELAPSED TIME= 00:00:00
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 67. Example output from a QUIESCE job that establishes a quiesce point for a list of objects

Example 3: Establishing a quiesce point for a table space set. The following control statement specifies that QUIESCE is to establish a quiesce point for the indicated table space set. In this example, the table space set includes table space DSN8D81A.DSN8S81D and all table spaces that are referentially related to it. Run REPORT TABLESPACESET to obtain a list of table spaces that are referentially related. For more information about this option, see Chapter 27, “REPORT,” on page 525.

```
QUIESCE TABLESPACESET TABLESPACE DSN8D81A.DSN8S81D
```

Figure 68. shows the output that the preceding command produces.

```
DSNU000I   DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TSLQ.STEP1
DSNU050I   DSNUGUTC - QUIESCE TABLESPACESET TABLESPACE DSN8D81A.DSN8S81D
DSNU477I - DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACESET DSN8D81A.DSN8S81D
DSNU477I - DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D81A.DSN8S81D
DSNU477I - DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D81A.DSN8S81E
DSNU477I - DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D81A.PROJ
DSNU477I - DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D81A.ACT
DSNU477I - DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D81A.PROJACT
DSNU477I - DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D81A.EMPPROJA
DSNU477I - DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D81A.DSN8S1D
DSNU474I - DSNUQUIA - QUIESCE AT RBA 000000052708 AND AT LRSN 000000052708
DSNU475I   DSNUQUIB - QUIESCE UTILITY COMPLETE, ELAPSED TIME= 00:00:25
DSNU010I   DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 68. Example output from a QUIESCE job that establishes a quiesce point for a table space set

Example 4: Establishing a quiesce point without writing the changed pages to disk. In the following example, the control statement specifies that the QUIESCE utility is to establish a quiesce point for table space DSN8D81A.DSN8S81D, without writing the changed pages to disk. (The default is to write the changed pages to disk.) In this example, a quiesce point is established for COPY YES indexes, but not for COPY NO indexes. Note that QUIESCE jobs with the WRITE YES option, which is the default, process both COPY YES indexes and COPY NO indexes. For both QUIESCE WRITE YES jobs and QUIESCE WRITE NO jobs, the utility inserts a row in SYSIBM.SYSCOPY for each COPY YES index.

```
//STEP1    EXEC DSNUPROC,UID='IUIQU2UD.QUIESC2',
//            UTPROC='',SYSTEM='DSN'
//SYSIN    DD *
//DSNUPROC.SYSIN    DD  *
QUIESCE TABLESPACE DSN8D81A.DSN8S81D WRITE NO
//*
```

The preceding command produces the output that is shown in Figure 69. Notice that the COPY YES index EMPNOI is placed in informational COPY-pending (ICOPY) status:

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    DSNUGUTC -  QUIESCE TABLESPACE DSN8D81A.DSN8S81D WRITE NO
DSNU477I  = DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D81A.DSN8S81D
DSNU477I  = DSNUQUIA -    QUIESCE SUCCESSFUL FOR INDEXSPACE DSN8D81A.EMPNOI
DSNU474I  = DSNUQUIA - QUIESCE AT RBA 000004E892A3 AND AT LRSN 000004E892A3
DSNU568I  = DSNUGSRX - INDEX ADMF001.EMPNOI IS IN INFORMATIONAL COPY PENDING
DSNU475I    DSNUQUIB - QUIESCE UTILITY COMPLETE, ELAPSED TIME= 00:00:00
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 69. Example output from a QUIESCE job that establishes a quiesce point, without writing the changed pages to disk.

Chapter 22. REBUILD INDEX

The REBUILD INDEX utility reconstructs indexes or index partitions from the table that they reference.

For a diagram of REBUILD INDEX syntax and a description of available options, see "Syntax and options of the REBUILD INDEX control statement." For detailed guidance on running this utility, see "Instructions for running REBUILD INDEX" on page 341.

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:
- RECOVERDB privilege for the database
- STATS privilege for the database is required if the STATISTICS keyword is specified.
- DBADM or DBCTRL authority for the database
- SYSCTRL or SYSADM authority

To run REBUILD INDEX STATISTICS REPORT YES, you must use a privilege set that includes STATS privilege for the database and the SELECT privilege on the catalog tables and tables for which statistics are to be gathered. REBUILD INDEX STATISTICS REPORT ALL does not report values from tables that the user is not authorized to see.

Execution phases of REBUILD INDEX: The REBUILD INDEX utility operates in the following phases:

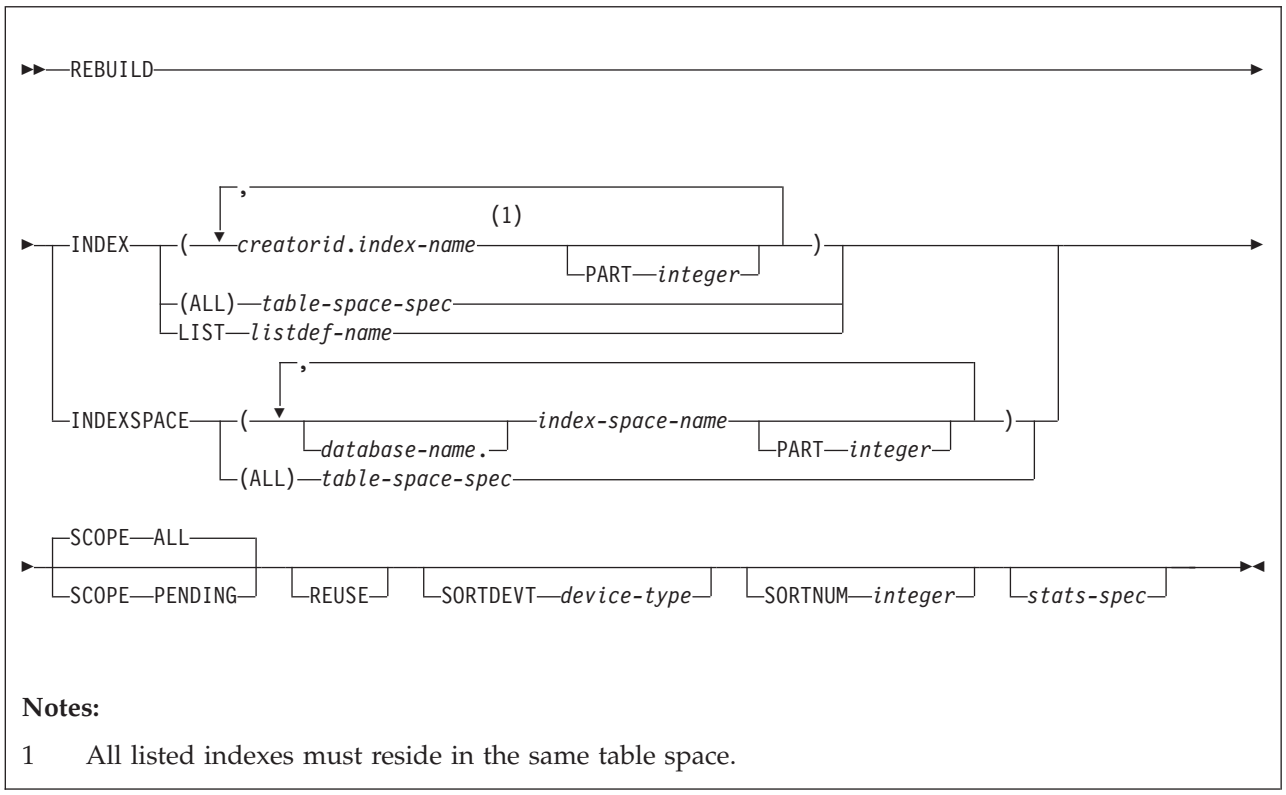
Phase	Description
UTILINIT	Performs initialization and setup.
UNLOAD	Unloads index entries.
SORT	Sorts unloaded index entries.
BUILD	Builds indexes.
SORTBLD	Sorts and builds a table space for parallel index build processing.
UTILTERM	Performs cleanup.

The following topics provide additional information:
- "Syntax and options of the REBUILD INDEX control statement"
- "Instructions for running REBUILD INDEX" on page 341
- "Concurrency and compatibility for REBUILD INDEX" on page 349
- "The effect of REBUILD INDEX on index version numbers" on page 350
- "Sample REBUILD INDEX control statements" on page 351

Syntax and options of the REBUILD INDEX control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

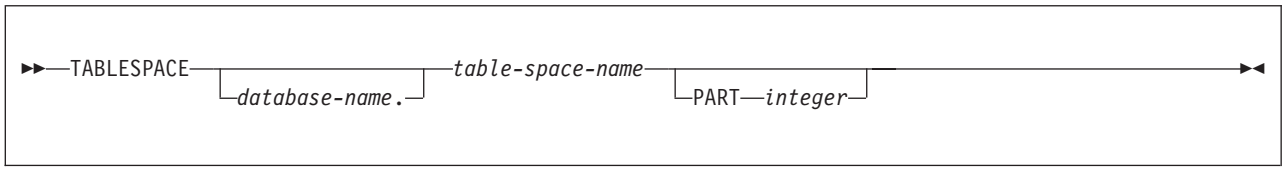
Syntax diagram



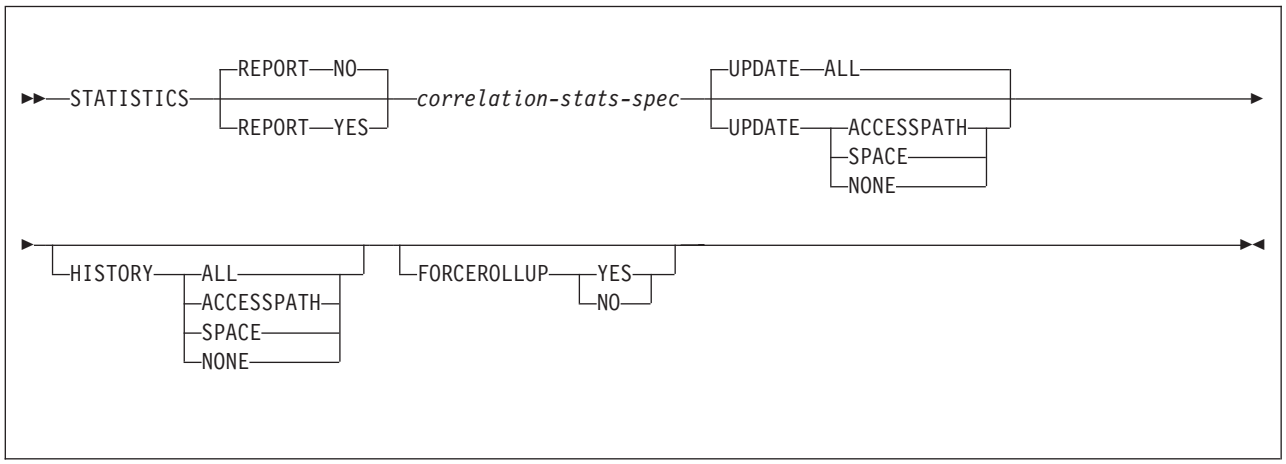
Notes:

1 All listed indexes must reside in the same table space.

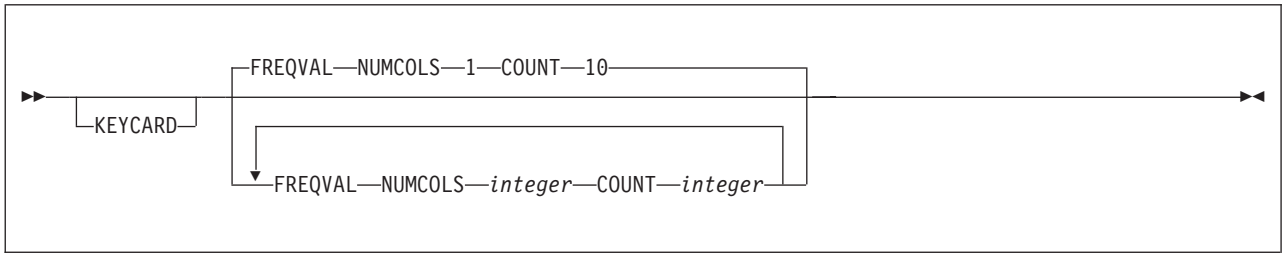
table-space-spec:



stats-spec:



correlation-stats-spec:



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

INDEX *creator-id.index-name*
Indicates the qualified name of the index to be rebuilt. Use the form *creator-id.index-name* to specify the name.

creator-id
Specifies the creator of the index. This qualifier is optional. If you omit the qualifier *creator-id*, DB2 uses the user identifier for the utility job.

index-name
Specifies the qualified name of the index that is to be rebuilt. For an index, you can specify either an index name or an index space name. Enclose the index name in quotation marks if the name contains a blank.

To rebuild multiple indexes, separate each index name with a comma. All listed indexes must reside in the same table space. If more than one index is listed and the TABLESPACE keyword is not specified, DB2 locates the first valid index name that is cited and determines the table space in which that index resides. That table space is used as the target table space for all other valid index names that are listed.

INDEXSPACE *database-name.index-space-name*
Specifies the qualified name of the index space that is obtained from the SYSIBM.SYSINDEXES table.

database-name
Specifies the name of the database that is associated with the index. This qualifier is optional.

index-space-name
Specifies the qualified name of the index space to copy. For an index, you can specify either an index name or an index space name.

If you specify more than one index space, they must all be defined on the same table space.

For an index, you can specify either an index name or an index space name.

(ALL)
Specifies that all indexes in the table space that is referred to by the TABLESPACE keyword are to be rebuilt.

TABLESPACE *database-name.table-space-name*
Specifies the table space from which all indexes are to be rebuilt.

database-name
Identifies the database to which the table space belongs. The **default** is **DSNDB04**.

table-space-name
Identifies the table space from which all indexes are to be rebuilt.

PART *integer*
Specifies the physical partition of a partitioning index or a data-partitioned secondary index in a partitioned table that is to be rebuilt. When the target of the REBUILD operation is a nonpartitioned secondary index, the utility reconstructs logical partitions. If any of the following situations are true for a nonpartitioned index, you cannot rebuild individual logical partitions:

- the index was created with DEFER YES
- the index must be completely rebuilt (This situation is likely in a disaster recovery scenario)
- the index is in page set REBUILD-pending (PSRBD) status

For these cases, you must rebuild the entire index.

integer is the number of the partition and must be in the range from 1 to the number of partitions that are defined for the table space. The maximum is 4096.

You cannot specify PART with the LIST keyword. Use LISTDEF PARTLEVEL instead.

LIST *listdef-name*
Specifies the name of a previously defined LISTDEF list name. The utility allows one LIST keyword for each REBUILD INDEX control statement. The list must contain either all index spaces or all table spaces. For a table space list, REBUILD is invoked once per table space. For an index space list, DB2 groups indexes by their related table space and executes the rebuild once per table space. For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

SCOPE
Indicates the scope of the rebuild organization of the specified index or indexes.

ALL
Indicates that you want the specified index or indexes to be rebuilt. The **default** is **ALL**.

PENDING
Indicates that you want the specified index or indexes with one or more partitions in REBUILD-pending (RBDP), REBUILD-pending star (RBDP*), page set REBUILD-pending (PSRBD), RECOVER-pending (RECP), or advisory REORG-pending (AREO*) state to be rebuilt.

REUSE
Specifies that REBUILD should logically reset and reuse DB2-managed data sets without deleting and redefining them. If you do not specify REUSE, DB2 deletes and redefines DB2-managed data sets to reset them.

If you are rebuilding the index because of a media failure, do not specify REUSE.

If a data set has multiple extents, the extents are not released if you use the REUSE parameter.

SORTDEVT *device-type*
Specifies the device type for temporary data sets that are to be dynamically allocated by DFSORT. For *device-type*, you can specify any device that is valid on the DYNALLOC parameter of the SORT or OPTION options for DFSORT. For more information about these options, see *DFSORT Application Programming: Guide*.

device-type is the device type.

A TEMPLATE specification does not dynamically allocate sort work data sets. The SORTDEVT keyword controls dynamic allocation of these data sets.

SORTNUM *integer*
Specifies the number of temporary data sets that are to be dynamically allocated by the sort program. If you omit SORTDEVT, SORTNUM is ignored. If you use SORTDEVT and omit SORTNUM, no value is passed to DFSORT; DFSORT uses its own default.

integer is the number of temporary data sets that can range from 2 to 255.

You need at least two sort work data sets for each sort. The SORTNUM value applies to each sort invocation in the utility. For example, if there are three indexes, SORTKEYS is specified, there are no constraints limiting parallelism, and SORTNUM is specified as 8, then a total of 24 sort work data sets will be allocated for a job.

Each sort work data set consumes both above the line and below the link virtual storage, so if you specify too high a value for SORTNUM, the utility may decrease the degree of parallelism due to virtual storage constraints, and possibly decreasing the degree down to one, meaning no parallelism.

Important: The SORTNUM keyword will not be considered if ZPARM UTSORTAL is set to YES and IGNSORTN is set to YES.

STATISTICS
Specifies that index statistics are to be collected.

If you specify the STATISTICS and UPDATE options, statistics are stored in the DB2 catalog. You cannot collect inline statistics for indexes on the catalog and directory tables.

Restriction: If you specify STATISTICS for encrypted data, DB2 might not provide useful statistics on this data. If the utility is terminated with the -TERM UTIL command after the STATISTICS have been updated in the catalog, the statistics are not rolled back. A subsequent RUNSTATS utility may be needed.

REPORT
Indicates whether a set of messages to report the collected statistics is to be generated.

NO
Indicates that the set of messages is not to be sent as output to SYSPRINT. The **default** is **NO**.

YES
Indicates that the set of messages is to be sent as output to SYSPRINT. The generated messages are dependent on the combination of keywords (such as TABLESPACE, INDEX, TABLE, and COLUMN) that you specify with the RUNSTATS utility. However, these messages are not dependent on the specification of the UPDATE option. REPORT YES always generates a report of SPACE and ACCESSPATH statistics.

KEYCARD
Specifies that all of the distinct values in all of the 1 to *n* key column combinations for the specified indexes are to be collected. *n* is the number of columns in the index.

FREQVAL
Controls the collection of frequent-value statistics. If you specify FREQVAL, it must be followed by two additional keywords:

NUMCOLS
Indicates the number of key columns that are to be concatenated when collecting frequent values from the specified index. If you specify 3, the utility collects frequent values on the concatenation of the first three key columns. The **default** is **1**, which means that DB2 is to collect frequent values only on the first key column of the index.

COUNT
Indicates the number of frequent values that are to be collected. If you specify 15, the utility collects 15 frequent values from the specified key columns. The **default** is **10**.

UPDATE
Indicates whether the collected statistics are to be inserted into the catalog tables. UPDATE also allows you to select statistics that are used for access path selection or statistics that are used by database administrators.

ALL Indicates that all collected statistics are to be updated in the catalog. The **default** is **ALL**.

ACCESSPATH
Indicates that the only catalog table columns that are to be updated are those that provide statistics that are used for access path selection.

SPACE
Indicates that the only catalog table columns that are to be updated are those that provide statistics to help the database administrator assess the status of a particular table space or index.

NONE
Indicates that catalog tables are not to be updated with the collected statistics. This option is valid only when REPORT YES is specified.

HISTORY
Records all catalog table inserts or updates to the catalog history tables.

The default is supplied by the value that is specified in STATISTICS HISTORY on panel DSNTIPO.

ALL Indicates that all collected statistics are to be updated in the catalog history tables.

ACCESSPATH
Indicates that the only catalog history table columns that are to be updated are those that provide statistics that are used for access path selection.

SPACE
Indicates that only space-related catalog statistics are to be updated in catalog history tables.

NONE
Indicates that catalog history tables are not to be updated with the collected statistics.

FORCEROLLUP
Specifies whether aggregation or rollup of statistics is to take place when you execute RUNSTATS even if some indexes or index partitions are empty. This keyword enables the optimizer to select the best access path.

The following options are available for the FORCEROLLUP keyword:

YES Indicates that forced aggregation or rollup processing is to be done, even though some indexes or index partitions might not contain data.

NO Indicates that aggregation or rollup is to be done only if data is available for all indexes or index partitions.

If data is not available, the utility issues DSNU623I message if you have set the installation value for STATISTICS ROLLUP on panel DSNTIPO to NO.

Instructions for running REBUILD INDEX

To run REBUILD INDEX, you must:

1. Read "Before running REBUILD INDEX" in this section.
2. Prepare the necessary data sets, as described in "Data sets that REBUILD INDEX uses."
3. Create JCL statements, by using one of the methods described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for REBUILD INDEX, see "Sample REBUILD INDEX control statements" on page 351.)
4. Prepare a utility control statement that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks" on page 343.
5. Check the compatibility table in "Concurrency and compatibility for REBUILD INDEX" on page 349 if you want to run other jobs concurrently on the same target objects.
6. Plan for restart if the REBUILD INDEX job doesn't complete, as described in "Terminating or restarting REBUILD INDEX" on page 349.
7. Run REBUILD INDEX by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Before running REBUILD INDEX

Because the data that DB2 needs to build an index is in the table space on which the index is based, you do not need image copies of indexes. To rebuild the index, you do not need to recover the table space, unless it is also damaged. You do not need to rebuild an index merely because you have recovered the table space on which it is based.

If you recover a table space to a prior point in time and do not recover all the indexes to the same point in time, you must rebuild all of the indexes.

Some logging might occur if both of the following conditions are true:

- The index is a nonpartitioning index.
- The index is being concurrently accessed either by SQL on a different partition of the same table space or by a utility that is run on a different partition of the same table space.

Data sets that REBUILD INDEX uses

Table 55 on page 342 lists the data sets that REBUILD INDEX uses. The table lists the DD name that is used to identify the data set, a description of the data set, and

an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 55. Data sets that REBUILD INDEX uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
STPRIN01	A data set that contains messages from DFSORT (usually, SYSOUT or DUMMY). This data set is used when frequency statistics are collected on DPSI's or when TABLESPACE TABLE COLGROUP FREQVAL is specified	No [1]
Work data sets	Temporary data sets for sort input and output when sorting keys If index build parallelism is used, the DD names have the form SW*nn*WK*mm*. If index build parallelism is not used, the DD names have the form SORTWK*nn*.	Yes
Sort work data sets	Temporary data sets for sort input and output when collecting inline statistics on at least one data-partitioned secondary index. The DD names have the form ST01WK*nn*.	No[1, 2, 3]

Notes:

1. Required when collecting inline statistics on at least one data-partitioned secondary index.
2. If the DYNALLOC parm of the SORT program is not turned on, you need to allocate the data set. Otherwise, DFSORT dynamically allocates the temporary data set.
3. It is recommended that you use dynamic allocation by specifying SORTDEVT in the utility statement because dynamic allocation reduces the maintenance required of the utility job JCL.

The following object is named in the utility control statement and does not require a DD statement in the JCL:

Table space
Object whose indexes are to be rebuilt.

Calculating the size of the work data sets: To calculate the approximate size (in bytes) of the SORTWK*nn* data set, use the following formula:

2 × (longest index key + c) × (number of extracted keys)

longest key The length of the longest index key that is to be processed by the subtask.

If the index is of varying length, the longest key is the maximum possible length of a key with all varying-length columns that are padded to their maximum length, plus 2 bytes for each varying-length column in the index. For example, if an index with 3 columns (A, B, and C) has length values of CHAR(8) for A, VARCHAR(128) for B, and VARCHAR(50) for C, the longest key is calculated as follows:

```
8 + 128 + 50 + 2 + 2 = 190
```

c A value as follows:
- 10 if the indexes that are being rebuilt are a mix of data-partitioned secondary indexes and nonpartitioned indexes
- 8 if the indexes that are being rebuilt are partitioned, or if none of them are data-partitioned secondary indexes.

number of keys The number of keys from all indexes that the subtask sorts and processes.

Using two or three large SORTWK*nn* data sets are preferable to several small ones.

Calculating the size of the sort work data sets: To calculate the approximate size (in bytes) of the ST01WK*nn* data set, use the following formula:

2 ×(*maximum record length* × *numcols* × (*count* + 2) × number of indexes)

The variables in the preceding formula have the following values:

maximum record length
Maximum record length of the SYSCOLDISTSTATS record that is processed when collecting frequency statistics (You can obtain this value from the RECLENGTH column in SYSTABLES.)

numcols
Number of key columns to concatenate when you collect frequent values from the specified index.

count Number of frequent values that DB2 is to collect.

DB2 utilities uses DFSORT to perform sorts. Sort work data sets cannot span volumes. Smaller volumes require more sort work data sets to sort the same amount of data; therefore, large volume sizes can reduce the number of needed sort work data sets. It is recommended that at least 1.2 times the amount of data to be sorted be provided in sort work data sets on disk. For more information about DFSORT, see *DFSORT Application Programming Guide*.

Creating the control statement

Create the utility control statement for the REBUILD INDEX job. See "Syntax and options of the REBUILD INDEX control statement" on page 335 for syntax and option descriptions. See "Sample REBUILD INDEX control statements" on page 351 for examples of usage.

Beginning in Version 8, the SORTKEYS option is the default. Therefore, the REBUILD INDEX utility does not require the SYSUT1 data set. The WORKDDN keyword, which provided the DD name of the SYSUT1 data set in earlier versions of DB2, is not needed and is ignored. The SORTKEYS keyword is also ignored. You do not need to modify existing control statements to remove the WORKDDN keyword or the SORTKEYS keyword.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in your utility control statement:

"Rebuilding index partitions" on page 344
"Improving performance when rebuilding index partitions" on page 344
"Building indexes in parallel" on page 344
"Resetting the REBUILD-pending status" on page 348

"Rebuilding critical catalog indexes" on page 348
"Recoverability of a rebuilt index" on page 348

Rebuilding index partitions

REBUILD INDEX can rebuild one or more partitions of a partitioned index by extracting the keys from the data rows of the table on which they are based. When you specify the PART option, one or more partitions from a partitioning index or a data-partitioned secondary index can be rebuilt. However, for nonpartitioned indexes, you cannot rebuild individual logical partitions in certain situations. See the description of the PART option (338) for more information about these situations.

Improving performance when rebuilding index partitions

If you use the PART option to rebuild only a single partition of an index, the utility does not need to scan the entire table space.

To rebuild several indexes (including data-partitioned secondary indexes) at the same time and reduce recovery time, use parallel index rebuild, or submit multiple index jobs. See "Building indexes in parallel" for more information.

When rebuilding nonpartitioned secondary indexes and partitions of partitioned indexes, this type of parallel processing on the same table space decreases the size of the sort data set, as well as the total time that is required to sort all the keys.

When you run the REBUILD INDEX utility concurrently on separate partitions of a partitioned index (either partitioning or secondary), the sum of the processor time is approximately the time for a single REBUILD INDEX job to run against the entire index. For partitioning indexes, the elapsed time for running concurrent REBUILD INDEX jobs is a fraction of the elapsed time for running a single REBUILD INDEX job against an entire index.

Building indexes in parallel

Parallel index build reduces the elapsed time for a REBUILD INDEX job by sorting the index keys and rebuilding multiple indexes or index partitions in parallel, rather than sequentially. Optimally, a pair of subtasks processes each index; one subtask sorts extracted keys, while the other subtask builds the index. REBUILD INDEX begins building each index as soon as the corresponding sort generates its first sorted record. If you specify STATISTICS, a third subtask collects the sorted keys and updates the catalog table in parallel.

The subtasks that are used for the parallel REBUILD INDEX processing use DB2 connections. If you receive message DSNU397I that indicates that the REBUILD INDEX utility is constrained, increase the number of concurrent connections by using the MAX BATCH CONNECT parameter on panel DSNTIPE.

The greatest elapsed processing-time improvements result from parallel rebuilding for:

- Multiple indexes on a table space
- A partitioning index or a data-partitioned secondary index on all partitions of a partitioned table space
- A nonpartitioned secondary index on a partitioned table space

Figure 70 on page 345 shows the flow of a REBUILD INDEX job with a parallel index build. The same flow applies whether you rebuild a data-partitioned secondary index or a partitioning index. DB2 starts multiple subtasks to unload the

entire partitioned table space. Subtasks then sort index keys and build the partitioning index in parallel. If you specify STATISTICS, additional subtasks collect the sorted keys and update the catalog table in parallel, eliminating the need for a second scan of the index by a separate RUNSTATS job.

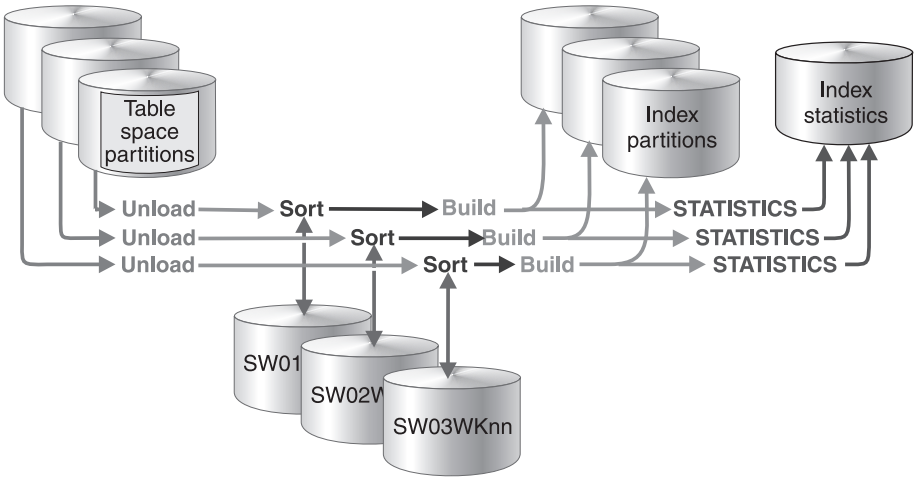


Figure 70. How a partitioning index is rebuilt during a parallel index build

Figure 71 shows the flow of a REBUILD INDEX job with a parallel index build. DB2 starts multiple subtasks to unload all partitions of a partitioned table space and to sort index keys in parallel. The keys are then merged and passed to the build subtask, which builds the nonpartitioned secondary index. If you specify STATISTICS, a separate subtask collects the sorted keys and updates the catalog table.

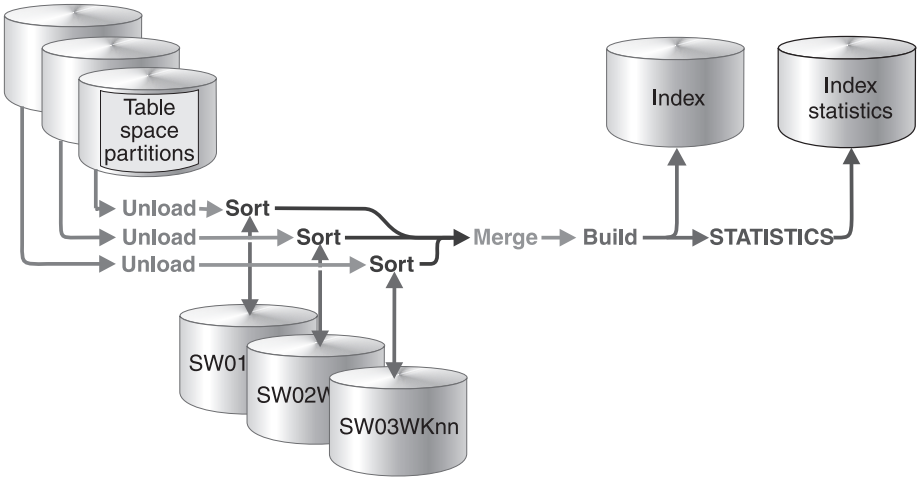


Figure 71. How a nonpartitioned secondary index is rebuilt during a parallel index build

When parallel index build is used: Beginning in Version 8, REBUILD INDEX always sorts the index keys and builds them in parallel for partitioned table spaces unless constrained by available memory, sort work files, or UTPRIN*nn* file allocations.

Sort work data sets for parallel index build: You can either allow the utility to dynamically allocate the data sets that SORT needs, or provide the necessary data sets yourself. Select one of the following methods to allocate sort work data sets and message data sets:

Method 1: REBUILD INDEX determines the optimal number of sort work data sets and message data sets.

1. Specify the SORTDEVT keyword in the utility statement.
2. Allow dynamic allocation of sort work data sets by **not** supplying SORTWK*nn* DD statements in the REBUILD INDEX utility JCL.
3. Allocate UTPRINT to SYSOUT.

Method 2: You control allocation of sort work data sets, and REBUILD INDEX allocates message data sets.

1. Provide DD statements with DD names in the form SW*nn*WK*mm*.
2. Allocate UTPRINT to SYSOUT.

Method 3: You have the most control over rebuild processing; you must specify both sort work data sets and message data sets.

1. Provide DD statements with DD names in the form SW*nn*WK*mm*.
2. Provide DD statements with DD names in the form UTPRIN*nn*.

Data sets that are used: If you select Method 2 or 3, define the necessary data sets by using the information provided here and in the following topics:

- "Determining the number of sort subtasks" on page 347
- "Allocation of sort subtasks" on page 347
- "Estimating the sort work file size" on page 347

Each sort subtask must have its own group of sort work data sets and its own print message data set. In addition, you need to allocate the merge message data set when you build a single nonpartitioned secondary index on a partitioned table space.

Possible reasons to allocate data sets in the utility job JCL rather than using dynamic allocation are to:

- Control the size and placement of the data sets
- Minimize device contention
- Optimally utilize free disk space
- Limit the number of utility subtasks that are used to build indexes

The DD names SW*nn*WK*mm* define the sort work data sets that are used during utility processing. *nn* identifies the subtask pair, and *mm* identifies one or more data sets that are to be used by that subtask pair. For example:

SW01WK01 Is the first sort work data set that is used by the subtask that builds the first index.

SW01WK02 Is the second sort work data set that is used by the subtask that builds the first index.

SW02WK01 Is the first sort work data set that is used by the subtask that builds the second index.

SW02WK02 Is the second sort work data set that is used by the subtask that builds the second index.

The DD names UTPRIN*nn* define the sort work message data sets that are used by the utility subtask pairs. *nn* identifies the subtask pair.

If you allocate the UTPRINT DD statement to SYSOUT in the job statement, the sort message data sets and the merge message data set, if required, are dynamically allocated. If you want the sort message data sets, merge message data sets, or both, allocated to a disk or tape data set rather than to SYSOUT, you must supply the UTPRIN*nn* or the UTMERG01 DD statements (or both) in the utility JCL. If you do not allocate the UTPRINT DD statement to SYSOUT, and you do not supply a UTMERG01 DD statement in the job statement, partitions are not unloaded in parallel.

Determining the number of sort subtasks: The maximum number of utility subtasks that are started for parallel index build equals:

- For a simple table space, segmented table space, or simple partition of a partitioned table space, the number of indexes that are to be built
- For a single index that is being built on a partitioned table space, the number of partitions that are to be unloaded

REBUILD INDEX determines the number of subtasks according to the following guidelines:

- The number of subtasks equals the number of allocated sort work data set groups.
- The number of subtasks equals the number of allocated message data sets.
- If you allocate both sort work data sets and message data set groups, the number of subtasks equals the smallest number of allocated data sets.

Allocation of sort subtasks: REBUILD INDEX attempts to assign one sort subtask for each index that is to be built. If REBUILD INDEX cannot start enough subtasks to build one index per subtask, it allocates any excess indexes across the pairs (in the order that the indexes were created), so that one or more subtasks might build more than one index.

Estimating the sort work file size: If you choose to provide the data sets, you need to know the size and number of keys that are present in all of the indexes or index partitions that are being processed by the subtask in order to calculate each sort work file size. When you determine which indexes or index partitions are assigned to which subtask pairs, use the formula listed in "Data sets that REBUILD INDEX uses" on page 341 to calculate the required space.

Overriding dynamic DFSORT allocation: DB2 estimates how many rows are to be sorted and passes this information to DFSORT on the parameter FILSZ. DFSORT then dynamically allocates the necessary sort work space.

If the table space contains rows with VARCHAR columns, DB2 might not be able to accurately estimate the number of rows. If the estimated number of rows is too high and the sort work space is not available or if the estimated number of rows is too low, DFSORT might fail and cause an abend. **Important:** Run RUNSTATS UPDATE SPACE before the REBUILD INDEX utility so that DB2 calculates a more accurate estimate.

You can override this dynamic allocation of sort work space in two ways:

- Allocate the sort work data sets with SORTWK*nn* DD statements in your JCL.
- Override the DB2 row estimate in FILSZ using control statements that are passed to DFSORT. However, using control statements overrides size estimates

that are passed to DFSORT in all invocations of DFSORT in the job step, including any sorts that are done in any other utility that is executed in the same step. The result might be reduced sort efficiency or an abend due to an out-of-space condition.

Resetting the REBUILD-pending status

REBUILD-pending status (which appears as RBDP in the output from the DISPLAY command) means that the physical or logical index partition, nonpartitioned secondary index, or logical partition of a nonpartitioned secondary index is in REBUILD-pending status.

The variations of REBUILD-pending status are as follows:

RBDP The physical or logical index partition is in the REBUILD-pending status. The individual physical or logical index partition is inaccessible. Reset the RBDP status by rebuilding the single affected partition. If multiple partitions are in RBDP status, you can rebuild either the entire index or *all* affected partitions.

RBDP*
The logical partition of the nonpartitioned secondary index is in the REBUILD-pending status. The entire nonpartitioned secondary index is inaccessible. Reset RBDP* status by rebuilding only the affected logical partitions.

PSRBD
The nonpartitioned secondary index space is in the REBUILD-pending status. The entire index space is inaccessible. Rebuild the object with the REBUILD INDEX utility. This state only applies to nonpartitioned secondary indexes.

You can reset the REBUILD-pending status for an index with any of these operations:
- REBUILD INDEX
- REORG TABLESPACE SORTDATA
- REPAIR SET INDEX with NORBDPEND
- START DATABASE command with ACCESS FORCE

Important: Use the START DATABASE command with ACCESS FORCE only as a means of last resort.

Rebuilding critical catalog indexes

If an ID with a granted authority tries to rebuild indexes in the catalog or directory, and if the DSNDB06.SYSDBASE or DSNDB06.SYSUSER table space is unavailable, you receive the following message:

```
DSNT501I, RESOURCE UNAVAILABLE
```

You must either make these table spaces available, or run the RECOVER TABLESPACE utility on the catalog or directory, using an authorization ID with the installation SYSADM or installation SYSOPR authority.

Recoverability of a rebuilt index

When you successfully rebuild an index that was defined with COPY YES, utility processing inserts a SYSCOPY row with ICTYPE='B' for each rebuilt index. Rebuilt indexes are also placed in informational COPY-pending status, which indicates that you should make a copy of the index.

Recommendation: Make a full image copy of the index to create a recovery point; this action also resets the ICOPY status.

Terminating or restarting REBUILD INDEX

You can terminate REBUILD INDEX by using the TERM UTILITY command. If you terminate a REBUILD INDEX job, the index space is placed in the REBUILD-pending status and is unavailable until it is successfully rebuilt.

By default, DB2 uses RESTART(PHASE) when restarting REBUILD INDEX jobs. The job starts again from the beginning.

If you restart a job that uses the STATISTICS keyword, inline statistics collection does not occur. To update catalog statistics, run the RUNSTATS utility after the restarted REBUILD INDEX job completes.

For more guidance about restarting online utilities, see "Restarting an online utility" on page 43.

Concurrency and compatibility for REBUILD INDEX

DB2 treats individual data and index partitions as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible.

Table 56 shows which claim classes REBUILD INDEX drains and any restrictive state that the utility sets on the target object.

Table 56. Claim classes of REBUILD INDEX operations.

Target	REBUILD INDEX	REBUILD INDEX PART
Table space or partition	DW/UTRO	DW/UTRO
Partitioning index, data-partitioned secondary index, or physical partition	DA/UTUT	DA/UTUT
Nonpartitioned secondary index	DA/UTUT	DR
Logical partition of an index	N/A	DA/UTUT

Legend:
- DA - Drain all claim classes; no concurrent SQL access
- DW - Drain the write claim class; concurrent access for SQL readers
- DR - Drains the repeatable-read claim class
- N/A - Not applicable
- UTUT - Utility restrictive state; exclusive control
- UTRO - Utility restrictive state; read-only access allowed

Table 57 shows which utilities can run concurrently with REBUILD INDEX on the same target object. The target object can be an index space or a partition of an index space. If compatibility depends on particular options of a utility, that information is also shown. REBUILD INDEX does not set a utility restrictive state if the target object is DSNDB01.SYSUTILX.

Table 57. Compatibility of REBUILD INDEX with other utilities

Action	REBUILD INDEX
CHECK DATA	No
CHECK INDEX	No

Table 57. Compatibility of REBUILD INDEX with other utilities (continued)

Action	REBUILD INDEX
CHECK LOB	Yes
COPY INDEX	No
COPY TABLESPACE SHRLEVEL CHANGE	No
COPY TABLESPACE SHRLEVEL REFERENCE	Yes
DIAGNOSE	Yes
LOAD	No
MERGECOPY	Yes
MODIFY	Yes
QUIESCE	No
REBUILD INDEX	No
RECOVER INDEX	No
RECOVER TABLESPACE	No
REORG INDEX	No
REORG TABLESPACE UNLOAD CONTINUE or PAUSE	No
REORG TABLESPACE UNLOAD ONLY or EXTERNAL with cluster index	No
REORG TABLESPACE UNLOAD ONLY or EXTERNAL without cluster index	Yes
REPAIR LOCATE by KEY	No
REPAIR LOCATE by RID DELETE or REPLACE	No
REPAIR LOCATE by RID DUMP or VERIFY	Yes
REPAIR LOCATE INDEX PAGE DUMP or VERIFY	No
REPAIR LOCATE TABLESPACE or INDEX PAGE REPLACE	No
REPAIR LOCATE TABLESPACE PAGE DUMP or VERIFY	Yes
REPORT	Yes
RUNSTATS INDEX	No
RUNSTATS TABLESPACE	Yes
STOSPACE	Yes
UNLOAD	Yes

To run REBUILD INDEX on SYSIBM.DSNLUX01 or SYSIBM.DSNLUX02, ensure that REBUILD INDEX is the only utility in the job step and the only utility that is running in the DB2 subsystem.

The effect of REBUILD INDEX on index version numbers

DB2 stores the range of used index version numbers in the OLDEST_VERSION and CURRENT_VERSION columns of the following catalog tables:

- SYSIBM.SYSINDEXES
- SYSIBM.SYSINDEXPART

The OLDEST_VERSION column contains the oldest used version number, and the CURRENT_VERSION column contains the current version number.

When you run REBUILD INDEX, the utility updates this range of used version numbers for indexes that are defined with the COPY NO attribute. REBUILD INDEX sets the OLDEST_VERSION column to the current version number, which indicates that only one version is active; DB2 can then reuse all of the other version numbers.

Recycling of version numbers is required when all of the version numbers are being used. All version numbers are being used when one of the following situations is true:

- The value in the CURRENT_VERSION column is one less than the value in the OLDEST_VERSION column
- The value in the CURRENT_VERSION column is 15, and the value in the OLDEST_VERSION column is 0 or 1.

You can also run LOAD REPLACE, REORG INDEX, or REORG TABLESPACE to recycle version numbers for indexes that are defined with the COPY NO attribute. To recycle version numbers for indexes that are defined with the COPY YES attribute or for table spaces, run MODIFY RECOVERY.

For more information about versions and how they are used by DB2, see Part 2 of *DB2 Administration Guide*.

Sample REBUILD INDEX control statements

Example 1: Rebuilding an index. The following control statement specifies that the REBUILD INDEX utility is to rebuild the DSN8810.XDEPT1 index.

```
//STEP1    EXEC DSNUPROC,UID='IUIQU2UT.RBLD1',TIME=1440,
//         UTPROC='',
//         SYSTEM='DSN',DB2LEV=DB2A
//SYSREC   DD DSN=IUIQU2UT.RBLD1.STEP1.SYSREC,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(8000,(20,20),,,ROUND)
//SYSIN    DD *
REBUILD INDEX (DSN8810.XDEPT1)
/*
```

Example 2: Rebuilding index partitions. The following control statement specifies that REBUILD INDEX is to rebuild partitions 2 and 3 of the DSN8810.XEMP1 index. The partition numbers are indicated by the PART option.

```
REBUILD INDEX (DSN8810.XEMP1 PART 2, DSN8810.XEMP1 PART 3)
```

Example 3: Rebuilding multiple partitions of a partitioning or secondary index. The following control statement specifies that REBUILD INDEX is to rebuild partitions 2 and 3 of the DSN8810.XEMP1 index. The partition numbers are indicated by the PART option. The SORTDEVT and SORTNUM keywords indicate that the utility is to use dynamic data set and message set allocation. Parallelism is used by default.

If sufficient virtual storage resources are available, DB2 starts one pair of utility sort subtasks for each partition. This example does not require UTPRIN*nn* DD statements because it uses DSNUPROC to invoke utility processing. DSNUPROC includes a DD statement that allocates UTPRINT to SYSOUT.

```
//SAMPJOB  JOB  ...
//STEP1    EXEC DSNUPROC,UID='SAMPJOB.RBINDEX',UTPROC='',SYSTEM='DSN'
//SYSIN    DD *
```

```
REBUILD INDEX (DSN8810.XEMP1 PART 2, DSN8810.XEMP1 PART 3)
 SORTDEVT SYSWK
 SORTNUM  4
/*
```

Example 4: Rebuilding all partitions of a partitioning index. The control statement in Figure 72 specifies that REBUILD INDEX is to rebuild all index partitions of the DSN8810.XEMP1 partitioning index. Parallelism is used by default. For this example, REBUILD INDEX allocates sort work data sets in two groups, which limits the number of utility subtask pairs to two. This example does not require UTPRIN*nn* DD statements because it uses DSNUPROC to invoke utility processing. DSNUPROC includes a DD statement that allocates UTPRINT to SYSOUT.

```
//SAMPJOB  JOB  ...
//STEP1    EXEC DSNUPROC,UID='SAMPJOB.RCVINDEX',UTPROC='',SYSTEM='DSN'
//* First group of sort work data sets for parallel index rebuild
//SW01WK01 DD UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SW01WK02 DD UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SW01WK03 DD UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//* Second group of sort work data sets for parallel index rebuild
//SW02WK01 DD UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SW02WK02 DD UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SW02WK03 DD UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SYSIN    DD *
  REBUILD INDEX (DSN8810.XEMP1)
/*
```

Figure 72. Example REBUILD INDEX statement

Example 5: Rebuilding all indexes of a table space. The following control statement specifies that REBUILD INDEX is to rebuild all indexes for table space DSN8D81A.DSN8S81E. The SORTDEVT and SORTNUM keywords indicate that the utility is to use dynamic data set and message set allocation. Parallelism is used by default.

If sufficient virtual storage resources are available, DB2 starts one utility sort subtask to build the partitioning index and another utility sort subtask to build the nonpartitioning index. This example does not require UTPRIN*nn* DD statements because it uses DSNUPROC to invoke utility processing. DSNUPROC includes a DD statement that allocates UTPRINT to SYSOUT.

```
//SAMPJOB  JOB  ...
//STEP1    EXEC DSNUPROC,UID='SAMPJOB.RCVINDEX',UTPROC='',SYSTEM='DSN'
//SYSIN    DD *
REBUILD INDEX (ALL) TABLESPACE DSN8D81A.DSN8S81E
 SORTDEVT SYSWK
 SORTNUM  4
/*
```

Example 6: Rebuilding indexes only if they are in a restrictive state and gathering inline statistics. The control statement in Figure 73 on page 353 specifies that REBUILD INDEX is to rebuild partition 9 of index ID0S482D if it is in REBUILD-pending (RBDP), RECOVER-pending (RECP), or advisory REORG-pending (AREO*) state. This condition that the index be in a certain restrictive state is indicated by the SCOPE PENDING option. The STATISTICS FORCEROLLUP YES option indicates that the utility is to collect inline statistics on the index partition that it is rebuilding and to force aggregation of those statistics.

```
//STEP6    EXEC DSNUPROC,UID='JUOSU248.CHK6',
//         UTPROC='',
//         SYSTEM='SSTR'
//UTPRINT  DD  SYSOUT=*
//SYSREC   DD DSN=JUOSU248.CHKIXPX.STEP6.SYSREC,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSCOPY  DD DSN=JUOSU248.CHKIXPX.STEP6.SYSCOPY,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT  DD DSN=JUOSU248.CHKIXPX.STEP6.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
    REBUILD INDEX (IDOS482D PART 9)
            STATISTICS FORCEROLLUP YES
            SCOPE PENDING
/*
```

Figure 73. Example REBUILD INDEX statement with STATISTICS option

Chapter 23. RECOVER

The online RECOVER utility recovers data to the current state or to a previous point in time by restoring a copy and then applying log records.

The largest unit of data recovery is the table space or index space; the smallest is the page. You can recover a single object, or a list of objects. The RECOVER utility recovers an entire table space, index space, a partition or data set, pages within an error range, or a single page. You recover data from image copies of an object and from log records that contain changes to the object. If the most recent full image copy data set is unusable, and previous image copy data sets exist in the system, RECOVER uses the previous image copy data sets.

For a diagram of RECOVER syntax and a description of available options, see "Syntax and options of the RECOVER control statement" on page 356. For detailed guidance on running this utility, see "Instructions for running RECOVER" on page 363.

Output: Output from RECOVER consists of recovered data (a table space, index, partition or data set, error range, or page within a table space).

If you specify the TOLOGPOINT, TORBA, TOCOPY, TOLASTCOPY, or TOLASTFULLCOPY option to recover data to a point in time, RECOVER puts any associated index spaces in REBUILD-pending status. You must run REBUILD INDEX to remove the REBUILD-pending status from the index space.

If you use the RECOVER utility to partially recover a referentially related table space set or a base table space and LOB table space set, you must ensure that you recover the entire set of table spaces. This task includes rebuilding or recovering all indexes (including indexes on auxiliary tables for a base table space and LOB table space set) to a common quiesce point or to a SHRLEVEL REFERENCE copy. If you do not include every member of the set, or if you do not recover the entire set to the same point in time, RECOVER sets the CHECK-pending status on for all dependent table spaces, base table spaces, or LOB table spaces in the set.

Recommendation: If you use the RECOVER utility to partially recover data and all indexes on the data, recover these objects to a common quiesce point or to a SHRLEVEL REFERENCE copy. Otherwise, RECOVER places all indexes in the CHECK-pending status.

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:
- RECOVERDB privilege for the database
- DBADM or DBCTRL authority for the database
- SYSCTRL or SYSADM authority

An ID with installation SYSOPR authority can also execute RECOVER, but only on a table space in the DSNDB01 or DSNDB06 database.

Restrictions on running RECOVER: RECOVER cannot recover a table space or index space that is defined to use a storage group that is defined with mixed specific and nonspecific volume IDs. If you specify such a table space or index space, the job terminates and you receive error message DSNU419I.

RECOVER cannot recover a point-in-time RECOVER INDEX if the recovery point precedes the first ALTER INDEX in Version 8 new-function mode that created a new index version.

Execution phases of RECOVER: The RECOVER utility operates in these phases:

Phase	Description
UTILINIT	Performs initialization and setup.
RESTORE	Locates and merges any appropriate image copies and restores the table space to a backup level; processes a list of objects in parallel if you specify the PARALLEL keyword.
RESTORER	If you specify the PARALLEL keyword, reads and merges the image copies.
RESTOREW	If you specify the PARALLEL keyword, writes the pages to the object.
LOGAPPLY	Applies any outstanding log changes to the object that is restored from the previous phase or step.
UTILTERM	Performs cleanup.

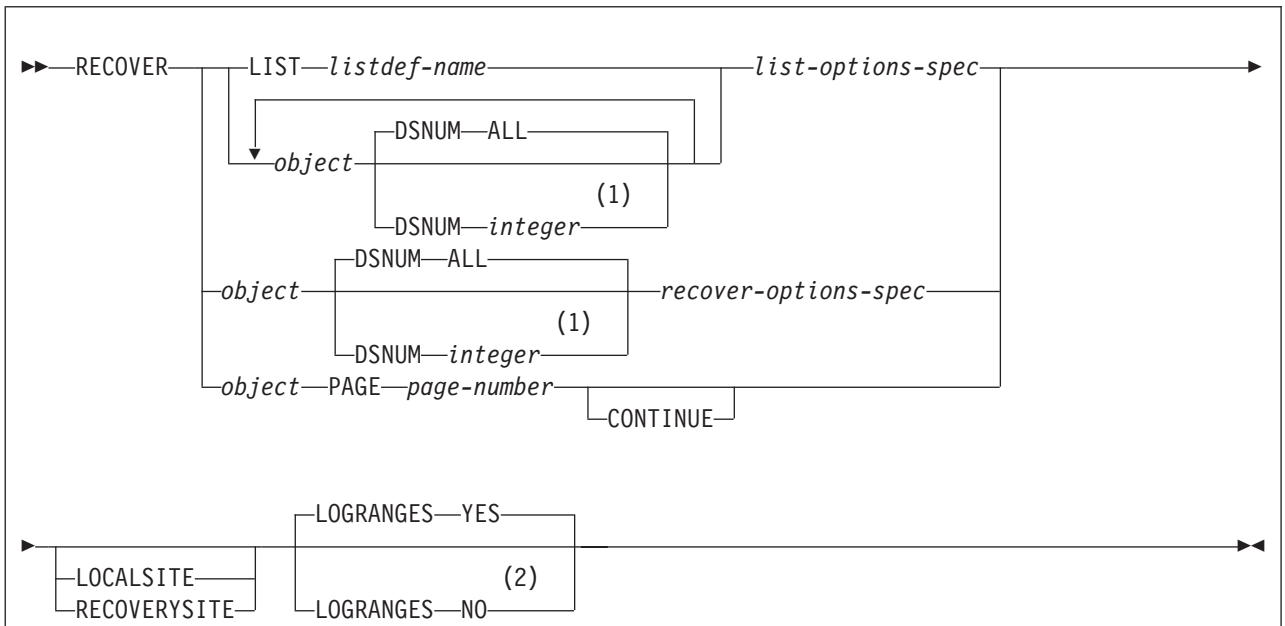
The following topics provide additional information:

- "Syntax and options of the RECOVER control statement"
- "Instructions for running RECOVER" on page 363
- "Terminating or restarting RECOVER" on page 384
- "Concurrency and compatibility for RECOVER" on page 384
- "Effects of running RECOVER" on page 386
- "Sample RECOVER control statements" on page 386

Syntax and options of the RECOVER control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram

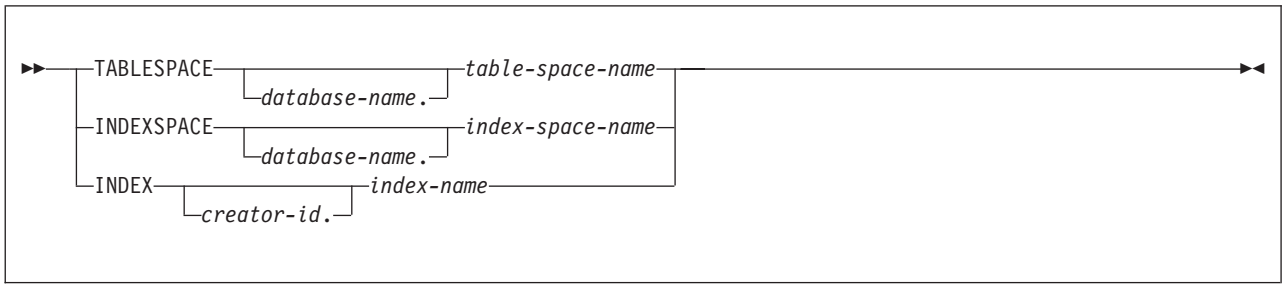


Notes:

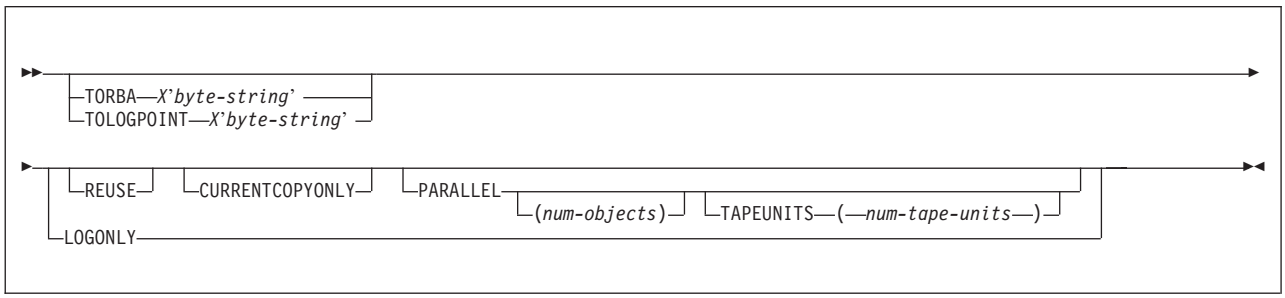
1 Not valid for nonpartitioning indexes.

2 Use the LOGRANGES NO option only at the direction of IBM Software Support. This option can cause the LOGAPPLY phase to run much longer and, in some cases, apply log records that should not be applied.

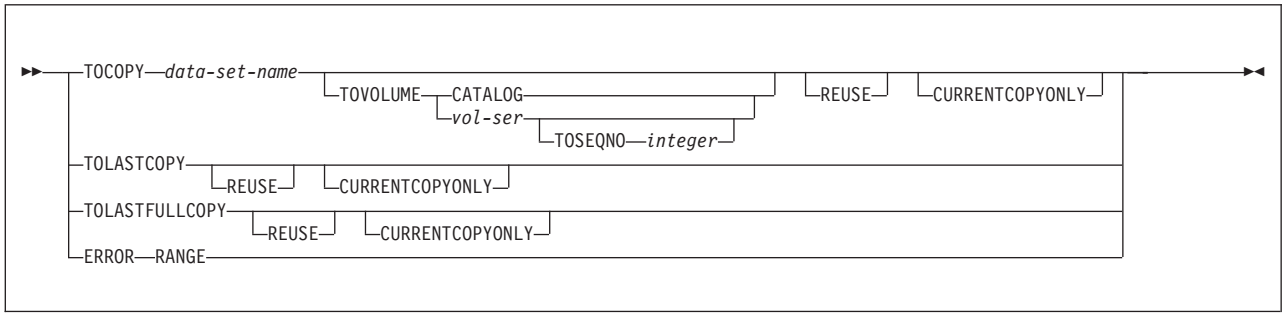
object:



list-options-spec:



recover-options-spec:



Option descriptions

You can specify a list of objects by repeating the TABLESPACE, INDEX, or INDEXSPACE keywords. If you use a list of objects, the valid keywords are: DSNUM, TORBA, TOLOGPOINT, LOGONLY, PARALLEL, and either LOCALSITE or RECOVERYSITE.

The options TOCOPY, TOLASTCOPY, TOLASTFULLCOPY, TORBA and TOLOGPOINT are all referred to as point-in-time recovery options.

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

LIST *listdef-name*
Specifies the name of a previously defined LISTDEF list name. The utility allows one LIST keyword for each control statement of RECOVER. The list can contain a mixture of table spaces and index spaces. RECOVER is invoked once for the entire list.

For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

TABLESPACE *database-name.table-space-name*
Specifies the table space (and optionally, the database to which it belongs) that is to be recovered.

You can specify a list of table spaces by repeating the TABLESPACE keyword. You can recover an individual catalog or directory table space in a list with its IBM-defined indexes. You cannot recover multiple catalog or directory table spaces in a list.

database-name Is the name of the database to which the table space belongs. The **default** is **DSNDB04**.

table-space-name
Is the name of the table space that is to be recovered.

INDEXSPACE *database-name.index-space-name*
Specifies the index space that is to be recovered.

database-name Specifies the name of the database to which the index space belongs. The **default** is **DSNDB04**.

index-space-name
Specifies the name of the index space that is to be recovered.

INDEX *creator-id.index-name*
Specifies the index in the index space that is to be recovered. The RECOVER utility can recover only indexes that were defined with the COPY YES attribute and subsequently copied.

creator-id
Optionally specifies the creator of the index. The **default** is the user identifier for the utility.

index-name
Specifies the name of the index in the index space that is to be recovered. Enclose the index name in quotation marks if the name contains a blank.

The following keywords are optional:

DSNUM
Identifies a partition within a partitioned table space or a partitioned index, or identifies a data set within a nonpartitioned table space that is to be recovered. You cannot specify a single data set of a nonpartitioned index or a logical partition of a nonpartitioned index. Alternatively, the option can recover the entire table space or index space.

ALL
Specifies that the entire table space or index space is to be recovered. The **default** is **ALL**.

integer
Specifies the number of the partition or data set that is to be recovered. The maximum value is 4096.

Specifying DSNUM is not valid for nonpartitioning indexes.

For a partitioned table space or index space: The integer is its partition number.

For a nonpartitioned table space: Find the integer at the end of the data set name. The data set name has the following format:

```
catname.DSNDBx.dbname.tsname.y0001.Annn
```

where:

catname	Is the VSAM catalog name or alias.
x	Is C or D.
dbname	Is the database name.
tsname	Is the table space name.
y	Is I or J.
nnn	Is the data set integer.

PAGE *page-number*
Specifies a particular page that is to be recovered. You cannot specify this option if you are recovering from a concurrent copy.

page-number is the number of the page, in either decimal or hexadecimal notation. For example, both 999 and X'3E7' represent the same page. PAGE is invalid with the LIST specification.

CONTINUE
Specifies that the recovery process is to continue. Use this option only if an error causes RECOVER to terminate during reconstruction of a page. In

this case, the page is marked as "broken". After you repair the page, you can use the CONTINUE option to recover the page, starting from the point of failure in the recovery log.

TORBA X'*byte-string***'**
Specifies, in a non-data-sharing environment, a point on the log to which RECOVER is to recover. Specify an RBA value.

In a data sharing environment, use TORBA only when you want to recover to a point before the originating member joined the data sharing group. If you specify an RBA after this point, the recovery fails.

Using TORBA terminates the recovery process with the last log record whose relative byte address (RBA) is not greater than *byte-string*, which is a string of up to 12 hexadecimal characters. If *byte-string* is the RBA of the first byte of a log record, that record is included in the recovery.

TOLOGPOINT X'*byte-string***'**
Specifies a point on the log to which RECOVER is to recover. Specify either an RBA or an LRSN value.

The LRSN is a string of 12 hexadecimal characters and is reported by the DSN1LOGP utility.

REUSE
Specifies that RECOVER is to logically reset and reuse DB2-managed data sets without deleting and redefining them. If you do not specify REUSE, DB2 deletes and redefines DB2-managed data sets to reset them.

If you are recovering an object because of a media failure, do not specify REUSE.

If a data set has multiple extents, the extents are not released if you use the REUSE parameter.

CURRENTCOPYONLY
Specifies that RECOVER is to improve the performance of restoring concurrent copies (copies that were made by the COPY utility with the CONCURRENT option) by using only the most recent primary copy for each object in the list.

When you specify CURRENTCOPYONLY for a concurrent copy, RECOVER builds a DFSMSdss RESTORE command for each group of objects that is associated with a concurrent copy data set name. If the RESTORE fails, RECOVER does not automatically use the next most recent copy or the backup copy, and the object fails. If you specify DSNUM ALL with CURRENTCOPYONLY and one partition fails during the restore process, the entire utility job on that object fails.

If you specify CURRENTCOPYONLY and the most recent primary copy of the object to be recovered is not a concurrent copy, DB2 ignores this keyword.

PARALLEL
Specifies the maximum number of objects in the list that are to be restored in parallel from image copies on disk or tape. RECOVER attempts to retain tape mounts for tapes that contain stacked image copies when the PARALLEL keyword is specified. In addition, to maximize performance, RECOVER determines the order in which objects are to be restored. If you specify TAPEUNITS with PARALLEL, you control the number of tape drives that are dynamically allocated for the recovery function. The TAPEUNITS keyword applies only to tape drives that are dynamically allocated. The TAPEUNITS keyword does not apply to JCL-allocated tape drives. The total number of tape

drives that are allocated for the RECOVER job is the sum of the JCL-allocated tape drives, and the number of tape drives, which is determined as follows:

- The specified value for TAPEUNITS.
- The value that is determined by the RECOVER utility if you omit the TAPEUNITS keyword. The number of tape drives that RECOVER attempts to allocate is determined by the object in the list that requires the most tape drives.

If you specify PARALLEL, you cannot specify TOCOPY, TOLASTCOPY, or TOLASTFULLCOPY.

(*num-objects***)**
Specifies the number of objects in the list that are to be processed in parallel. If storage constraints are encountered, you can adjust this value to a smaller value.

If you specify 0 or do not specify a value for *num-objects*, RECOVER determines the optimal number of objects to process in parallel.

TAPEUNITS
Specifies the number of tape drives that the utility should dynamically allocate for the list of objects that are to be processed in parallel. If you omit this keyword, the utility determines the number of tape drives to allocate for the recovery function.

(*num-tape-units***)**
Specifies the number of tape drives to allocate. If you specify 0 or do not specify the TAPEUNITS keyword, RECOVER determines the maximum number of tape units to use at one time.

LOGONLY
Specifies that the target objects are to be recovered from their existing data sets by applying only log records to the data sets. DB2 applies all log records that were written after a point that is recorded in the data set itself.

To recover an index space by using RECOVER LOGONLY, you must define the index space with the COPY YES attribute.

Use the LOGONLY option when the data sets of the target objects have already been restored to a point of consistency by another process offline, such as DFSMSdss concurrent copy.

TOCOPY *data-set-name*
Specifies the particular image copy data set that DB2 is to use as a source for recovery.

data-set-name is the name of the data set.

If the data set is a full image copy, it is the only data set that is used in the recovery. If it is an incremental image copy, RECOVER also uses the previous full image copy and any intervening incremental image copies.

If you specify the data set as the local backup copy, DB2 first tries to allocate the local primary copy. If the local primary copy is unavailable, DB2 uses the local backup copy.

If you use TOCOPY or TORBA to recover a single data set of a nonpartitioned table space, DB2 issues message DSNU520I to warn that the table space can become inconsistent following the RECOVER job. This point-in-time recovery can cause compressed data to exist without a dictionary or can even overwrite the data set that contains the current dictionary.

If you use TOCOPY with a particular partition or data set (identified with DSNUM), the image copy must be for the same partition or data set, or for the whole table space or index space. If you use TOCOPY with DSNUM ALL, the image copy must be for DSNUM ALL. You cannot specify TOCOPY with a LIST specification.

If the image copy data set is a z/OS generation data set, supply a fully qualified data set name, including the absolute generation and version number.

If the image copy data set is not a generation data set and more than one image copy data set with the same data set name exists, use one of the following options to identify the data set exactly:

TOVOLUME
Identifies the image copy data set. You cannot specify the TOVOLUME option with a LIST specification.

CATALOG
Indicates that the data set is cataloged. Use this option only for an image copy that was created as a cataloged data set. (Its volume serial is not recorded in SYSIBM.SYSCOPY.)

RECOVER refers to the SYSIBM.SYSCOPY catalog table during execution. If you use TOVOLUME CATALOG, the data set must be cataloged. If you remove the data set from the catalog after creating it, you must catalog the data set again to make it consistent with the record for this copy that appears in SYSIBM.SYSCOPY.

vol-ser
Identifies the data set by an alphanumeric volume serial identifier of its first volume. Use this option only for an image copy that was created as a noncataloged data set. Specify the first *vol-ser* in the SYSCOPY record to locate a data set that is stored on multiple tape volumes.

TOSEQNO *integer*
Identifies the image copy data set by its file sequence number. *integer* is the file sequence number.

TOLASTCOPY
Specifies that RECOVER is to restore the object to the last image copy that was taken. If the last image copy is a full image copy, it is restored to the object. If the last image copy is an incremental image copy, the most recent full copy along with any incremental copies are restored to the object. You cannot specify the TOLASTCOPY option with a LIST specification.

TOLASTFULLCOPY
Specifies that the RECOVER utility is to restore the object to the last full image copy that was taken. Any incremental image copies that were taken after the full image copy are not restored to the object. You cannot specify the TOLASTFULLCOPY option with a LIST specification.

ERROR RANGE
Specifies that all pages within the range of reported I/O errors are to be recovered. Recovering an error range is useful when the range is small, relative to the object that contains it; otherwise, recovering the entire object is
| preferred. You cannot specify this option if you are recovering from a
| concurrent copy.

In some situations, recovery using the ERROR RANGE option is not possible, such as when a sufficient quantity of alternate tracks cannot be obtained for all bad records within the error range. You can use the IBM Device Support

Facility, ICKDSF service utility to determine whether this situation exists. In such a situation, redefine the error data set at a different location on the volume or on a different volume, and then run the RECOVER utility without the ERROR RANGE option.

You cannot specify ERROR RANGE with a LIST specification.

For additional information about the use of this keyword, see Part 4 (Volume 1) of *DB2 Administration Guide*.

LOCALSITE
Specifies that RECOVER is to use image copies from the local site. If you specify neither LOCALSITE or RECOVERYSITE, RECOVER uses image copies from the current site of invocation. (The current site is identified on the installation panel DSNTIPO under SITE TYPE and in the macro DSN6SPRM under SITETYP.)

RECOVERYSITE
Specifies that RECOVER is to use image copies from the recovery site. If you specify neither LOCALSITE or RECOVERYSITE, RECOVER uses image copies from the current site of invocation. (The current site is identified on the installation panel DSNTIPO under SITE TYPE and in the macro DSN6SPRM under SITETYP.)

LOGRANGES YES
Specifies that RECOVER should use SYSLGRNX information for the LOGAPPLY phase. This option is the default.

LOGRANGES NO
Specifies that RECOVER should not use SYSLGRNX information for the LOGAPPLY phase. Use this option only under the direction of IBM Software Support.

This option can cause RECOVER to run much longer. In a data sharing environment this option can result in the merging of all logs from all members that were created since the last image copy.

This option can also cause RECOVER to apply logs that should not be applied. For example, assume that you take an image copy of a table space and then run REORG LOG YES on the same table space. Assume also that the REORG utility abends and you then issue the TERM UTILITY command for the REORG job. The SYSLGRNX records that are associated with the REORG job are deleted, so a RECOVER job with the LOGRANGES YES option (the default) skips the log records from the REORG job. However, if you run RECOVER LOGRANGES NO, the utility applies these log records.

Instructions for running RECOVER

To run RECOVER, you must:

1. Read "Before running RECOVER" on page 364 in this section.
2. Prepare the necessary data sets, as described in "Data sets that RECOVER uses" on page 364.
3. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.
4. Prepare a utility control statement that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks" on page 365.
5. Check the compatibility table in "Concurrency and compatibility for RECOVER" on page 384 if you want to run other jobs concurrently on the same target objects.

6. Plan for restart if the RECOVER utility job does not complete, as described in "Terminating or restarting RECOVER" on page 384.
7. Run RECOVER by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Before running RECOVER

Recovering data and indexes: You do not always need to recover both the data and indexes. If you recover the table space or index space to a current RBA or LRSN, any referentially related objects do not need to be recovered. If you plan to recover a damaged object to a point in time, ensure that you use a consistent point in time for all of its referentially related objects, including related LOB table spaces. You must rebuild the indexes from the data if one of the following conditions is true:

- The table space is recovered to a point in time.
- An index is damaged.
- An index is in REBUILD-pending status.
- No image copy of the index is available.

If you need to recover both the data and the indexes, and no image copies of the indexes are available, use the following procedure:

1. Use RECOVER TABLESPACE to recover the data.
2. Run REBUILD INDEX on any related indexes to rebuild them from the data.

If you have image copies of both the table spaces and the indexes, you can recover both sets of objects in the same RECOVER utility statement. The objects are recovered from the image copies and logs.

If the table space or index space to be recovered is associated with a storage group, DB2 deletes and redefines the necessary data sets. If the STOGROUP has been altered to remove the volume on which the table space or index space is located, RECOVER places the data set on another volume of the storage group.

Data sets that RECOVER uses

Table 58 lists the data sets that RECOVER uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 58. Data sets that RECOVER uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes

The following objects are named in the utility control statement and do not require DD statements in the JCL:

Table space or index space Object that is to be recovered. If you want to recover less than an entire table space:

- Use the DSNUM option to recover a partition or data set.
- Use the PAGE option to recover a single page.
- Use the ERROR RANGE option to recover a range of pages with I/O errors.

Image copy data set Copy that RECOVER is to restore. DB2 accesses this information through the DB2 catalog. However, if you want to preallocate your image copy data sets by using DD statements, refer to "Retaining tape mounts" on page 383 for more information.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in your utility control statement:

Recovering a table space

The following RECOVER statement specifies that the utility is to recover table space DSN8S81D in database DSN8D81A:

```
RECOVER TABLESPACE DSN8D81A.DSN8S81D
```

To recover multiple table spaces, create a list of table spaces that are to be recovered; repeat the TABLESPACE keyword before each specified table space. The following RECOVER statement specifies that the utility is to recover partition 2 of the partitioned table space DSN8D81A.DSN8S81E, and recover the table space DSN8D81A.DSN8S81D to the quiesce point (RBA X'000007425468').

```
RECOVER TABLESPACE DSN8D81A.DSN8S81E DSNUM 2
        TABLESPACE DSN8D81A.DSN8S81D
        TORBA X'000007425468'
```

Each table space that is involved is unavailable for most other applications until recovery is complete. If you make image copies by table space, you can recover the entire table space, or you can recover a data set or partition from the table space. If you make image copies separately by partition or data set, you must recover the partitions or data sets by running separate RECOVER operations. The following example shows the RECOVER statement for recovering four data sets in database DSN8D81A, table space DSN8S81E:

```
RECOVER TABLESPACE DSN8D81A.DSN8S81E DSNUM 1
        TABLESPACE DSN8D81A.DSN8S81E DSNUM 2
        TABLESPACE DSN8D81A.DSN8S81E DSNUM 3
        TABLESPACE DSN8D81A.DSN8S81E DSNUM 4
```

You can schedule the recovery of these data sets in four separate jobs to run in parallel. In many cases, the four jobs can read the log data concurrently.

If a table space or data set is in the COPY-pending status, recovering it might not be possible. You can reset this status in several ways; for more information, see "Resetting COPY-pending status" on page 269.

Recovering a list of objects

You can recover any of the following objects:

- Table space
- Table space partition
- Piece of a linear table space
- Index space
- Index space partition

When you recover an object to a prior point in time, you should recover a set of referentially related table spaces together to avoid putting any of the table spaces in CHECK-pending status. Use REPORT TABLESPACESET to obtain a table space listing.

RECOVER does not place dependent table spaces that are related by informational referential constraints into CHECK-pending status.

The RECOVER utility merges incremental copies serially and dynamically. As a result, recovery of a table space list with numerous incremental copies can be time-consuming and operator-intensive.

If referential integrity violations are not an issue, you can run a separate job to recover each table space.

When you specify the PARALLEL keyword, DB2 supports parallelism during the RESTORE phase and performs recovery as follows:

- During initialization and setup (the UTILINIT recover phase), the utility locates the full and incremental copy information for each object in the list from SYSIBM.SYSCOPY.
- The utility sorts the list of objects for recovery into lists to be processed in parallel according to the number of tape volumes, file sequence numbers, and sizes of each image copy.
- The number of objects that can be restored in parallel depends on the maximum number of available tape devices and on how many tape devices the utility requires for the incremental and full image copy data sets. You can control the number of objects that are to be processed in parallel on the PARALLEL keyword. You can control the number of dynamically allocated tape drives on the TAPEUNITS keyword, which is specified with the PARALLEL keyword.
- If an object in the list requires a DB2 concurrent copy, the utility sorts the object in its own list and processes the list in the main task, while the objects in the other sorted lists are restored in parallel. If the concurrent copies that are to be restored are on tape volumes, the utility uses one tape device and counts it toward the maximum value that is specified for TAPEUNITS.

Recovering a data set or partition

You can use the RECOVER utility to recover individual partitions and data sets. The phases for data set recovery are the same as for table space recovery.

If image copies are taken at the data set level, RECOVER must be performed at the data set level. To recover the whole table space, you must recover all the data sets individually in one or more RECOVER steps. If recovery is attempted at the table space level, DB2 returns an error message.

Alternatively, if image copies are taken at the table space, index, or index space level, you can recover individual data sets by using the DSNUM parameter.

RECOVER does **not** support recovery of the following types of indexes:

- A single data set for nonpartitioned secondary indexes
- A logical partition of a nonpartitioned secondary index

Recovering with incremental copies

The RECOVER utility merges all incremental image copies that were taken since the last full image copy. The utility must have all the image copies available at the same time. If this requirement is likely to strain your system resources, for example, by demanding more tape units than are available, consider running MERGECOPY regularly to merge image copies into one copy.

Even if you do not periodically merge multiple image copies into one copy when you do not have enough tape units, the utility can still perform. RECOVER dynamically allocates the full image copy and attempts to dynamically allocate all the incremental image copy data sets. If RECOVER successfully allocates every incremental copy, recovery proceeds to merge pages to table spaces and apply the log. If a point is reached where an incremental copy cannot be allocated, RECOVER notes the log RBA or LRSN of the last successfully allocated data set. Attempts to allocate incremental copies cease, and the merge proceeds using only the allocated data sets. The log is applied from the noted RBA or LRSN, and the incremental image copies that were not allocated are ignored.

Recovering a page

Using RECOVER PAGE enables you to recover data on a page that is damaged. In some situations, you can determine (usually from an error message) which page of an object has been damaged. You can use the PAGE option to recover a single page. You can use the CONTINUE option to continue recovering a page that was damaged during the LOGAPPLY phase of a RECOVER operation.

Recovering a page by using PAGE and CONTINUE: Suppose that you start RECOVER for table space TSPACE1. During processing, message DSNI012I informs you of a problem that damages page number 5. RECOVER completes, but the damaged page, number 5, is in a stopped state and is not recovered. When RECOVER ends, message DSNU501I informs you that page 5 is damaged.

To repair the damaged page:

1. Use the DUMP option of the REPAIR utility to view the contents of the damaged page. Determine what change should have been made by the applicable log record, and apply it by using the REPLACE option of REPAIR. Use the RESET option to turn off the inconsistent-data indicator.

Attention: Be extremely careful when using the REPAIR utility to replace data. Using REPAIR to change data to invalid values can produce unpredictable results, particularly when you change page header information. Improper use of REPAIR can result in damaged data, or in some cases, system failure.

2. Resubmit the RECOVER utility job by specifying TABLESPACE(TSPACE1) PAGE(*5*) CONTINUE. The RECOVER utility finishes recovering the damaged page by applying the log records that remain after the one that caused the problem.

If more than one page is damaged during RECOVER, perform the preceding steps for each damaged page.

Recovering an error range

By using the ERROR RANGE option of RECOVER, you can repair pages with reported I/O errors. DB2 maintains a page error range for I/O errors for each data set; pages within the range cannot be accessed. The DISPLAY DATABASE command displays the range. When recovering an error range, RECOVER:

1. Locates, allocates, and applies image copies.
2. Applies changes from the log.

The following RECOVER statement specifies that the utility is to recover any current error range problems for table space TS1:

```
RECOVER TABLESPACE DB1.TS1 ERROR RANGE
```

Recovering an error range is useful when the range is small, relative to the object containing it; otherwise, recovering the entire object is preferable.

Message DSNU086I indicates that I/O errors were detected on a table space and that you need to recover it. Before you attempt to use the ERROR RANGE option of RECOVER, you should run the ICKDSF service utility to correct the disk error. If an I/O error is detected during RECOVER processing, DB2 issues message DSNU538I to identify the affected target tracks are involved. The message provides enough information to run ICKDSF correctly.

In some situations, which are announced by error messages, recovery of only an error range is not possible. In such a situation, recovering the entire object is preferable.

During the recovery of the entire table space or index space, DB2 might still encounter I/O errors that indicate DB2 is still using a bad volume. For user-defined data sets, you should use Access Method Services to delete the data sets and redefine them with the same name on a new volume. If you use DB2 storage groups, you can remove the bad volume from the storage group by using ALTER STOGROUP.

Recovering with a data set copy that is not made by DB2

You can restore a data set to a point of consistency by using a data set copy that was not made by the COPY utility. After recovery to the point of consistency, if you choose to continue and recover to the current point in time, you do not want RECOVER to begin processing by restoring the data set from a DB2 image copy. Therefore, use the LOGONLY option of RECOVER, which causes RECOVER to skip the RESTORE phase and apply the log records only, starting from the first log record that was written after the data set was backed up.

Because the data sets are restored offline without DB2 involvement, RECOVER LOGONLY checks that the data set identifiers match those that are in the DB2 catalog. If the identifiers do not match, message DSNU548I is issued, and the job terminates with return code 8.

You can use the LOGONLY option on a list of objects.

To ensure that no other transactions can access DB2 objects between the time that you restore a data set and the time that you run RECOVER LOGONLY, follow these steps:

1. Stop the DB2 objects that are being recovered by issuing the following command:

   ```
   -STOP DATABASE(database-name) SPACENAM(space-name)
   ```

2. Restore all DB2 data sets that are being recovered.
3. Start the DB2 objects that are being recovered by issuing the following command:

   ```
   -START DATABASE(database-name) SPACENAM(space-name) ACCESS(UT)
   ```

4. Run the RECOVER utility without the TORBA or TOLOGPOINT parameters and with the LOGONLY parameter to recover the DB2 data sets to the current point in time and to perform forward recovery using DB2 logs. If you want to recover the DB2 data sets to a prior point in time, run the RECOVER utility with either TORBA or TOLOGPOINT, and with the LOGONLY parameters.
5. If you did not recover related indexes in the same RECOVER control statement, rebuild all indexes on the recovered object.
6. Issue the following command to allow access to the recovered object if the recovery completes successfully:

   ```
   -START DATABASE(database-name) SPACENAM(space-name) ACCESS(RW)
   ```

With the LOGONLY option, when recovering a single piece of a multi-piece linear page set, RECOVER opens the first piece of the page set. If the data set is migrated by DFSMShsm, the data set is recalled by DFSMShsm. Without LOGONLY, no data set recall is requested.

Backing up a single piece of a multi-piece linear page set is not recommended. This action can cause a data integrity problem if the backup is used to restore the data set at a later time.

Recovering catalog and directory objects

If you are recovering any subset of the objects in the following list, start with the object that appears first, and continue in the order of the list. For example, if you need to recover SYSLGRNX, SYSUTILX, and SYSUSER, recover first SYSUTILX, then SYSLGRNX, and then SYSUSER. You do not need to recover all of the objects; only recover those objects that require recovery. If you copy your catalog or directory indexes, use the RECOVER utility to recover your indexes. Otherwise, use the REBUILD INDEX utility to rebuild those indexes.

1. DSNDB01.SYSUTILX.
2. All indexes on SYSUTILX.
3. DSNDB01.DBD01.
4. DSNDB06.SYSCOPY.
5. All indexes on SYSIBM.SYSCOPY. If no user-defined indexes that are stogroup-managed are defined on SYSIBM.SYSCOPY, execute the following utility statement to rebuild IBM-defined and any user-defined indexes on SYSIBM.SYSCOPY:

```
REBUILD INDEX (ALL) TABLESPACE DSNDB06.SYSCOPY
```

If user-defined indexes that are stogroup-managed are defined on SYSIBM.SYSCOPY, rebuild the IBM-defined indexes by name (REBUILD INDEX (SYSIBM.index-name-1, SYSIBM.index-name-2, . . ., SYSIBM.index-name-n)) and then rebuild the user-defined indexes in a subsequent step. See Appendix D of *DB2 SQL Reference* for a list of the IBM-defined indexes.

6. DSNDB01.SYSLGRNX.
7. All indexes on SYSLGRNX.
8. DSNDB06.SYSALTER.
9. All indexes on SYSALTER.
10. DSNDB06.SYSDBAUT.
11. All indexes on SYSDBAUT. If no user-defined indexes that are stogroup-managed are defined on SYSDBAUT, execute the following utility statement to rebuild IBM-defined and any user-defined indexes on SYSDBAUT:

```
REBUILD INDEX (ALL) TABLESPACE DSNDB06.SYSDBAUT
```

If user-defined indexes that are stogroup-managed are defined on SYSDBAUT, rebuild the IBM-defined indexes by name (REBUILD INDEX (SYSIBM.index-name-1, SYSIBM.index-name-2, . . ., SYSIBM.index-name-n)) and then rebuild the user-defined indexes in a subsequent step. See Appendix D of *DB2 SQL Reference* for a list of the IBM-defined indexes.

12. DSNDB06.SYSUSER.
13. DSNDB06.SYSDBASE.
14. All indexes on SYSDBASE and SYSUSER. If no user-defined indexes that are stogroup-managed are defined on SYSDBASE or SYSUSER, execute the following utility statement to rebuild IBM-defined and any user-defined indexes on SYSDBASE and SYSUSER:

```
REBUILD INDEX (ALL) TABLESPACE DSNDB06.SYSxxxx
```

If user-defined indexes that are stogroup managed are defined on SYSDBASE and SYSUSER, rebuild the IBM-defined indexes by name (REBUILD INDEX (SYSIBM.index-name-1, SYSIBM.index-name-2, . . ., SYSIBM.index-name-n)), and then rebuild the user-defined indexes in a subsequent step. See Appendix D of *DB2 SQL Reference* for a list of the IBM-defined indexes.

15. Other catalog and directory table spaces and their IBM-defined indexes. The remaining catalog table spaces, in database DSNDB06, are SYSGROUP, SYSGPAUT, SYSOBJ, SYSPLAN, SYSPKAGE, SYSSEQ, SYSSEQ2, SYSSTATS, SYSSTR, SYSVIEWS, SYSDDF, SYSHIST, SYSGRTNS, SYSJAVA, SYSJAUXA, SYSJAUXB, and SYSEBCDC. The two remaining directory table spaces are DSNDB01.SCT02, which has index SYSIBM.DSNSCT02, and DSNDB01.SPT01, which has indexes SYSIBM.DSNSPT01 and SYSIBM.DSNSPT02.

    Note: At the start of the covesion for enabling-new-function mode, catalog indes DSNDB06.DSNKCX01 is dropped. In enabling-new-function mode or new-function mode, this index no longer needs to be rebuilt or recovered since it does not exist.

16. All user-defined indexes on the catalog that have not been rebuilt or recovered yet.
17. System utility table spaces, such as DB2 QMF.

18. If used, real-time statistics objects, the object and application registration tables, and the resource limit specification tables.
19. User table spaces.

For all catalog and directory table spaces, you can list the IBM-defined indexes that have the COPY YES attribute in the same RECOVER utility statement.

The catalog and directory objects that are listed in step 15 in the preceding list 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, the utility needs to make only one pass of the log for all objects during the LOGAPPLY phase and can use parallelism when restoring the image copies in the RESTORE phase. Thus, these objects are recovered faster.

Recovery of the items on the list can be done concurrently or included in the same job step. However, some restrictions apply:

1. When you recover the following table spaces or indexes, the job step in which the RECOVER statement appears must not contain any other utility statements. No other utilities can run while the RECOVER utility is running.
   - DSNDB01.SYSUTILX
   - All indexes on SYSUTILX
   - DSNDB01.DBD01
2. When you recover the following table spaces, no other utilities can run while the RECOVER utility is running. Other utility statements can exist in the same job step.
   - DSNDB06.SYSCOPY
   - DSNDB01.SYSLGRNX
   - DSNDB06.SYSDBAUT
   - DSNDB06.SYSUSER
   - DSNDB06.SYSDBASE

If the logging environment requires adding or restoring active logs, restoring archive logs, or performing any action that affects the log inventory in the BSDS, you should recover the BSDS before catalog and directory objects. For information about recovering the BSDS, see Part 4 (Volume 1) of *DB2 Administration Guide*. To copy active log data sets, use the Access Method Services REPRO function. For information about the JCL for the Access Method Services REPRO function, see one of the following publications:

- *DFSMS/MVS: Access Method Services for the Integrated Catalog*
- *z/OS DFSMS Access Method Services for Catalogs*

Why the order is important: To recover one object, RECOVER must obtain information about it from some other object. Table 59 lists the objects from which RECOVER must obtain information.

Table 59. Objects that the RECOVER utility accesses

Object name	Reason for access by RECOVER
DSNDB01.SYSUTILX	Utility restart information. The object is not accessed when it is recovered; RECOVER for this object is not restartable, and no other commands can be in the same job step. SYSCOPY information for SYSUTILX is obtained from the log.

Table 59. Objects that the RECOVER utility accesses (continued)

Object name	Reason for access by RECOVER
DSNDB01.DBD01	Descriptors for the catalog database (DSNDB06), the work file database (DSNDB07), and user databases. RECOVER for this object is not restartable, and no other commands can be in the same job step. SYSCOPY information for DBD01 is obtained from the log.
DSNDB06.SYSCOPY	Locations of image copy data sets. SYSCOPY information for SYSCOPY itself is obtained from the log.
DSNDB01.SYSLGRNX	The RBA or LRSN of the first log record after the most recent copy.
DSNDB06.SYSDBAUT, DSNDB06.SYSUSER	Verification that the authorization ID is authorized to run RECOVER.
DSNDB06.SYSDBASE	Information about table spaces that are to be recovered.

You can use REPORT RECOVERY to obtain SYSCOPY information for DSNDB01.SYSUTILX, DSNDB01.DBD01, and DSNDB06.SYSCOPY.

Planning for point-in-time recovery for the catalog and directory: When you recover the DB2 catalog and directory, consider the entire catalog and directory, including all table spaces and index spaces, as one logical unit. Recover all objects in the catalog and directory to the same point of consistency. If a point-in-time recovery of the catalog and directory objects is planned, a separate quiesce of the DSNDB06.SYSCOPY table space is required after a quiesce of the other catalog and directory table spaces.

You should be aware of some special considerations when you are recovering catalog and directory objects to a point in time in which the DB2 subsystem was in a different mode. For example, if your DB2 subsystem is currently in new-function mode, and you need to recover to a point in time in which the subsystem was in compatibility mode. For details, see Part 4 of *DB2 Administration Guide*.

Recommendation: Before you recover the DB2 catalog and directory objects to a prior point in time, shut down the DB2 system cleanly and then restart the system in access(maint) mode. Recover the catalog and directory objects to the current state. You can use sample queries and documentation, which are provided in DSNTESQ in the SDSNSAMP sample library, to check the consistency of the catalog.

Indexes are rebuilt by REBUILD INDEX. If the only items you have recovered are table spaces in the catalog or directory, you might need to rebuild their indexes. Use the CHECK INDEX utility to determine whether an index is inconsistent with the data it indexes. You can use the RECOVER utility to recover catalog and directory indexes if the index was defined with the COPY YES attribute and if you have a full index image copy.

You must recover the catalog and directory before recovering user table spaces.

Be aware that the following table spaces, along with their associated indexes, do not have entries in SYSIBM.SYSLGRNX, even if they were defined with COPY YES:

- DSNDB01.SYSUTILX
- DSNDB01.DBD01
- DSNDB01.SYSLGRNX
- DSNDB06.SYSCOPY
- DSNDB06.SYSGROUP
- DSNDB01.SCT02
- DSNDB01.SPT01

These objects are assumed to be open from the point of their last image copy, so the RECOVER utility processes the log from that point forward.

Point-in-time recovery: Full recovery of the catalog and directory table spaces and indexes is strongly recommended. However, if you need to plan for point-in-time recovery of the catalog and directory, here is a way to create a point of consistency:

1. Quiesce all catalog and directory table spaces in a list, except for DSNDB06.SYSCOPY and DSNDB01.SYSUTILX.
2. Quiesce DSNDB06.SYSCOPY.

   Recommendation: Quiesce DSNDB06.SYSCOPY in a separate utility statement; when you recover DSNDB06.SYSCOPY to its own quiesce point, it contains the ICTYPE = 'Q' (quiesce) SYSCOPY records for the other catalog and directory table spaces.
3. Quiesce DSNDB01.SYSUTILX in a separate job step.

If you need to recover to a point in time, recover DSNDB06.SYSCOPY and DSNDB01.SYSUTILX to their own quiesce points, and recover other catalog and directory table spaces to their common quiesce point. The catalog and directory objects must be recovered in a particular order, as described in "Why the order is important" on page 371.

Recovering critical catalog table spaces: An ID with a granted authority receives message `DSNT500I RESOURCE UNAVAILABLE` while trying to recover a table space in the catalog or directory if table space DSNDB06.SYSDBASE or DSNDB06.SYSUSER is unavailable. If you receive this message, you must either make these table spaces available, or run the RECOVER utility on the catalog or directory by using an authorization ID that has the installation SYSADM or installation SYSOPR authority.

Reinitializing DSNDB01.SYSUTILX

You need to reinitialize the DSNDB01.SYSUTILX directory table space if both of the following conditions are true:

- You cannot successfully execute the -DIS UTIL and -TERM UTIL commands, because DSNDB01.SYSUTILX is damaged.
- You cannot recover DSNDB01.SYSUTILX, because errors occur in the LOGAPPLY phase.

Because DSNDB01.SYSUTILX contains information about active and outstanding utilities, the process of reinitializing this table space involves determining which objects have a utility in progress and resolving any pending states to make the object available for access.

If DSNDB01.SYSUTILX must be reinitialized, use the following procedure with caution:

1. Issue the -DIS DB(*) SPACENAM(*) RESTRICT command and analyze the output. Write down the following items:
   - All of the objects with a utility in progress (The objects in UTUT, UTRO, or UTRW status have utilities in progress.)
   - Any pending states for these objects (RECP, CHKP, and COPY are examples of pending states. For a complete list, see Appendix C, "Advisory or restrictive states," on page 853.)
2. Edit the following installation jobs so that they contain only the commands that pertain to DSNDB01.SYSUTILX:

   DSNTIJDE
   Delete VSAM LDS for DSNDB01.SYSUTILX.

   DSNTIJIN
   Define VSAM LDS for DSNDB01.SYSUTILX and tailor the AMS DEFINE command to fit the needs of your DB2 system.

   DSNTIJID
   Initialize DSNDB01.SYSUTILX.
3. Run the three edited installation jobs in the order listed.
4. Issue the -START DB(*dbname*) ACCESS(UT) command for each database that has objects with a utility in progress.
5. Issue the -START DB(*dbname*)SPACENAM(*spname*) ACCESS(FORCE) command on each object with a utility in progress. This action clears all utilities that are in progress or in pending states. (Any pending states are cleared, but you still need to resolve the pending states as directed in the next step.)
6. Resolve the pending states for each object by running the appropriate utility. For example, if an object was in the RECP status, run the RECOVER utility. For more information about how to resolve pending states, see Appendix C, "Advisory or restrictive states," on page 853.
7. Issue -START DB(*dbname*) ACCESS(RW) for each database.

Recovering a table space that contains LOB data

The RECOVER utility can set the auxiliary warning status for a LOB table space if it finds at least one invalid LOB column. DB2 marks a LOB invalid if all of the following conditions are true:

1. The LOB table space was defined with the LOG(NO) attribute.
2. The LOB table space was recovered.
3. The LOB was updated since the last image copy.

The status of an object that is related to a LOB table space can change due to a recovery operation, depending on the type of recovery that is performed. If all of the following objects for all LOB columns are recovered in a single RECOVER utility statement to the present point in time, a QUIESCE point, or a COPY SHRLEVEL(REFERENCE) point, no pending status exists:

- Base table space
- Index on the auxiliary table
- LOB table space

Refer to Table 60 on page 375 for information about the status of a base table space, index on the auxiliary table, or LOB table space that was recovered without its related objects.

Table 60. Object status after being recovered without its related objects

Object	Recovery type	Base table space status	Index on the auxiliary table status	LOB table space status
Base table space	Current RBA or LRSN	None	None	None
Base table space	Point-in-time	CHECK- pending[1]	None	None
Index on the auxiliary table	Current RBA or LRSN	None	None	None
Index on the auxiliary table	Point-in-time	None	CHECK- pending[1]	None
LOB table space	Current RBA or LRSN, LOB table space that is defined with LOG(YES)	None	None	None
LOB table space	Current RBA or LRSN, LOB table space that is defined with LOG(NO)	None	None	Auxiliary warning[2]
LOB table space	TOCOPY, COPY was SHRLEVEL REFERENCE	CHECK- pending[1]	REBUILD- pending	None
LOB table space	TOCOPY, COPY was SHRLEVEL CHANGE	CHECK- pending[1]	REBUILD- pending	CHECK- pending or auxiliary warning[1]
LOB table space	TOLOGPOINT or TORBA (not a quiesce point)	CHECK- pending[1]	REBUILD- pending	CHECK- pending or auxiliary warning[1]
LOB table space	TOLOGPOINT or TORBA (at a quiesce point)	CHECK- pending[1]	REBUILD- pending	None

Notes:

1. RECOVER does not place dependent table spaces that are related by informational referential constraints into CHECK-pending status.
2. If, at any time, a log record is applied to the LOB table space and a LOB is consequently marked invalid, the LOB table space is set to auxiliary warning status.

For information about resetting any of these statuses, see Appendix C, "Advisory or restrictive states," on page 853.

Performing a point-in-time recovery

A recovery operation that is done with one of the point in time recovery options is known as a point-in-time recovery. A consistent point-in-time is a quiesce point or a set of image copies that was taken with SHRLEVEL REFERENCE. You do not need to take a full image copy after recovering to a point in time, except in the case of fallback recovery; see "Performing fallback recovery" on page 383. DB2 records the RBAs or LRSNs that are associated with the point-in-time recovery in the SYSIBM.SYSCOPY catalog table to allow future recover operations to skip the unwanted range of log records.

Because a point-in-time recovery of only the table space leaves data in a consistent state and indexes in an inconsistent state, you must rebuild all indexes by using REBUILD INDEX. For more information, see "Resetting the REBUILD-pending status" on page 348.

After an index has been altered to PADDED or NOT PADDED, you cannot recover that index to a prior point in time. Instead, you should rebuild the index.

If you use a point-in-time recovery option to recover a single data set of a nonpartitioned table space, DB2 issues message DSNU520I to warn that the table space can become inconsistent following the RECOVER job. This point-in-time recovery can cause compressed data to exist without a dictionary or can even overwrite the data set that contains the current dictionary.

The auxiliary CHECK-pending status (ACHKP) is set when the CHECK DATA utility detects an inconsistency between a base table space with defined LOB columns and a LOB table space. For information about how to reset the ACHKP status, see Appendix C, "Advisory or restrictive states," on page 853.

You can also use point-in-time recovery and the point-in-time recovery options to recover all user-defined table spaces and indexes that are in refresh-pending status (REFP).

For more information about recovering data to a prior point of consistency, see Part 4 (Volume 1) of *DB2 Administration Guide*.

Recovery considerations after rebalancing partitions with REORG: Image copies that were taken prior to resetting the REORG-pending status of any partition of a partitioned table space are not usable for recovering to a current RBA or LRSN. Avoid performing a point-in-time recovery for a partitioned table space to a point in time that is after the REORG-pending status was set, but before a rebalancing REORG was performed. To determine an appropriate point in time:

1. Run REPORT RECOVERY.
2. Select an image copy for which the recovery point is a point after the rebalancing REORG was performed.

If you run the REORG utility to turn off a REORG-pending status, and then recover to a point in time before that REORG job, DB2 sets restrictive statuses on all partitions that you specified in the REORG job, as follows:

- Sets REORG-pending (and possibly CHECK-pending) on for the data partitions
- Sets REBUILD-pending on for the associated index partitions
- Sets REBUILD-pending on for the associated logical partitions of nonpartitioned secondary indexes

For information about resetting these restrictive statuses, see "REORG-pending status" on page 858 and "REBUILD-pending status" on page 856.

Recommendation: To create a new recovery point, take one of these actions immediately following an ALTER INDEX or TABLE operation that changes partition boundaries:

- Run REORG with COPYDDN and SHRLEVEL NONE specified.
- Take a full image copy immediately after REORG completes.

Using offline copies to recover after rebalancing partitions: To recover data after a REORG job redistributes the data among partitions, use RECOVER LOGONLY. If you perform a point-in-time recovery, you must keep the offline copies synchronized with the SYSCOPY records. Therefore, do **not** use the MODIFY RECOVERY utility to delete any SYSCOPY records with an ICTYPE column value of 'A' because these records might be needed during the recovery. Delete these SYSCOPY records only when you are sure that you no longer need to use the offline copies that were taken before the REORG that performed the rebalancing.

Actions that can affect recovery status When you perform the following actions before you recover a table space, the recovery status is affected as described:

- If you alter a table to rotate partitions:
  - You can recover the partition to the current time.
  - You can recover the partition to a point in time after the alter. The utility can use a recovery base, (for example, a full image copy, a REORG LOG YES operation, or a LOAD REPLACE LOG YES operation) that occurred prior to the alter.
  - You cannot recover the partition to a point in time prior to the alter; the recover fails with MSGDSNU556I and RC8.
- If you change partition boundaries with ALTER or REORG REBALANCE:
  - You can recover the partition to the current time if a recovery base (for example, a full image copy, a REORG LOG YES operation, or a LOAD REPLACE LOG YES operation) exists.
  - You can recover the partition to a point in time after the alter.
  - You can recover the partitions that are affected by the boundary change to a point in time prior to the alter; RECOVER sets REORG-pending status on the affected partitions and you must reorganize the table space or range of partitions. All affected partitions must be in the recovery list of a single RECOVER statement.
- If you alter a table to add a partition:
  - You can recover the partition to the current time.
  - You can recover the partition to a point in time after the alter.
  - You can recover the partition to a point in time prior to the alter; RECOVER resets the partition to be empty.

When you perform the following actions before you recover an index to a prior point in time or to the current time, the recovery status is affected as described:

- If you alter the data type of a column to a numeric data type, you cannot recover the index until you take a full image copy of the index. However, the index can be rebuilt.
- If you alter an index to NOT PADDED or PADDED , you cannot recover the index until you take a full image copy of the index. However, the index can be rebuilt.

For information about recovery status, see Appendix C, "Advisory or restrictive states," on page 853.

Planning for point-in-time recovery: Point-in-time recovery options are viable alternatives in many situations in which recovery to the current point in time is not possible or desirable. To make these options work best for you, take periodic quiesce points at points of consistency that are appropriate to your applications.

When making copies of a single object, use SHRLEVEL REFERENCE to establish consistent points for TOCOPY, TOLASTCOPY, or TOLASTFULLCOPY recovery. Copies that are made with SHRLEVEL CHANGE do not copy data at a single instant because changes can occur as the copy is made. A subsequent RECOVER TOCOPY operation can produce inconsistent data.

When copying a list of objects, use SHRLEVEL REFERENCE. If a subsequent recovery to a point in time is necessary, you can use a single RECOVER utility statement to list all of the objects, along with TOLOGPOINT to identify the common RBA or LRSN value. If you use SHRLEVEL CHANGE to copy a list of objects, you should follow it with a QUIESCE of the objects.

To improve the performance of the recovery, take a full image copy of the table space or set of table spaces, and then quiesce them by using the QUIESCE utility. This action enables RECOVER TORBA to recover the table spaces to the quiesce point with minimal use of the log.

Authorization: Restrict use of the point-in-time recovery options to personnel with a thorough knowledge of the DB2 recovery environment.

Ensuring consistency: You can use RECOVER TORBA, RECOVER TOLOGPOINT, and RECOVER TOCOPY to recover one of the following single objects:

- Partition of a partitioned table space
- Partition of a partitioning index space
- Data set of a simple table space

For any of the previously listed objects, restore all data sets to the same level; otherwise, the data becomes inconsistent.

If possible, specify a table space and all of its indexes (or a set of table spaces and all related indexes) in the same RECOVER utility statement, and specify TOLOGPOINT or TORBA to identify a QUIESCE point. This action avoids placing indexes in the CHECK-pending or REBUILD-pending status. If the TOLOGPOINT is not a common QUIESCE point for all objects, use the following procedure:

1. RECOVER table spaces to the value for TOLOGPOINT (either an RBA or LRSN).
2. Use concurrent REBUILD INDEX jobs to recover the indexes over each table space.

This procedure ensures that the table spaces and indexes are synchronized, and it eliminates the need to run the CHECK INDEX utility.

Resetting CHECK-pending status: Point-in-time recovery can cause table spaces to be placed in CHECK-pending status if they have table check constraints or referential constraints defined on them. When recovering tables that are involved in a referential constraint, you should recover all the table spaces that are involved in a constraint.

RECOVER does not place dependent table spaces that are related by informational referential constraints into CHECK-pending status.

The TORBA and TOLOGPOINT options set the CHECK-pending status for table spaces when you perform any of the following actions:

- Recover one or more members of a set of table spaces to a previous point in time that is not a common quiesce point or SHRLEVEL(REFERENCE) point. Dependent table spaces are placed in CHECK-pending status.
- Recover all members of a set of table spaces that are to be recovered to the same quiesce point, but referential constraints were defined for a dependent table after that quiesce point. Table spaces that contain those dependent tables are placed in CHECK-pending status.
- Recover table spaces with defined LOB columns without recovering their LOB table spaces.

To avoid setting CHECK-pending status, you must perform both of the following steps:

- Recover the table space or the set of table spaces to a quiesce point or to an image copy that was made with SHRLEVEL REFERENCE.

If you do not recover each table space to the same quiesce point, and if any of the table spaces are part of a referential integrity structure, the following actions occur:

- All dependent table spaces that are recovered are placed in CHECK-pending status with the scope of the whole table space.
- All dependent table spaces of the recovered table spaces are placed in CHECK-pending status with the scope of the specific dependent tables.

- Do not add table check constraints or referential constraints after the quiesce point or image copy.

  If you recover each table space of a table space set to the same quiesce point, but referential constraints were defined after the quiesce point, the CHECK-pending status is set for the table space that contains the table with the referential constraint.

The TORBA and TOLOGPOINT options set the CHECK-pending status for indexes when you perform either of the following actions:

- Recover one or more of the indexes to a previous point in time, but you do not recover the related table space in the same RECOVER statement.
- Recover one or more of the indexes along with the related table space to a previous point in time that is not a quiesce point or SHRLEVEL REFERENCE point.

You can turn off CHECK-pending status for an index by using the TORBA and TOLOGPOINT options. Recover indexes along with the related table space to the same quiesce point or SHRLEVEL REFERENCE point. RECOVER processing resets the CHECK-pending status for all indexes in the same RECOVER statement.

For information about actions to take if CHECK INDEX identifies inconsistencies after you perform a RECOVER job, see "Reviewing CHECK INDEX output" on page 91.

For information about resetting the CHECK-pending status of table spaces, see Chapter 8, "CHECK DATA," on page 59. For information about resetting the CHECK-pending status for indexes, see "CHECK-pending status" on page 854.

Compressed data: Use caution when recovering a portion of a table space or partition (for example, one data set) to a prior point in time. If the data set that is being recovered has been compressed with a different dictionary, you can no longer read the data. The details of data compression are described in Part 5 (Volume 2) of *DB2 Administration Guide*.

Avoiding specific image copy data sets

You might accidentally lose an image copy, or you might want to avoid a specific image copy data set. Because the corresponding row is still present in SYSIBM.SYSCOPY, RECOVER always attempts to allocate the data set. This section describes the options that are available if you want to skip a specific image copy data set.

Image copy on tape: If the image copy is on tape, messages IEF233D and IEF455D request the tape for RECOVER, as shown in the following example:

```
 IEF233D M BAB,COPY   ,,R92341QJ,DSNUPROC,
         OR RESPOND TO IEF455D MESSAGE
*42 IEF455D MOUNT COPY    ON BAB FOR R92341QJ,DSNUPROC OR REPLY 'NO'
 R 42,NO
 IEF234E K BAB,COPY    ,PVT,R92341QJ,DSNUPROC
```

By replying NO, you can initiate the fallback to the previous image copy. RECOVER responds with messages DSNU030I and DSNU508I, as shown in the following example:

```
DSNU030I    csect-name - UNABLE TO ALLOCATE R92341Q.UTQPS001.FCOPY010
                         RC=4, CODE=X'04840000'
DSNU508I    csect-name - IN FALLBACK PROCESSING TO PRIOR FULL IMAGE COPY
```

Reason code X'0484' means that the request was denied by the operator.

Image copy on disk: If the image copy is on disk, you can delete or rename the image copy data set before RECOVER starts executing. RECOVER issues messages DSNU030I and DSNU508I, as shown in the following example:

```
DSNU030I    csect-name - UNABLE TO ALLOCATE R92341Q.UTQPS001.FCOPY010,
                         RC=4, CODE=X'17080000'
DSNU508I    csect-name - IN FALLBACK PROCESSING TO PRIOR FULL IMAGE COPY
```

Reason code X'1708' means that the ICF catalog entry cannot be found.

Improving performance

To improve recovery time, consider enabling the Fast Log Apply function on the DB2 subsystem. For more information about enabling this function, see the LOG APPLY STORAGE field on panel DSNTIPL, in Part 2 of *DB2 Installation Guide*.

Use MERGECOPY to merge your table space image copies before recovering the table space. If you do not merge your image copies, RECOVER automatically merges them. If RECOVER cannot allocate all the incremental image copy data sets when it merges the image copies, RECOVER uses the log instead.

Include a list of table spaces and indexes in your RECOVER utility statement to apply logs in a single scan of the logs.

If you use RECOVER TOCOPY for full image copies, you can improve performance by using data compression. The improvement is proportional to the degree of compression.

Consider specifying the PARALLEL keyword to restore image copies from disk or tape to a list of objects in parallel.

If you are recovering concurrent copies, consider specifying the CURRENTCOPYONLY option to improve performance. When you specify this option, RECOVER can issue one DFSMSdss RESTORE command for multiple objects. The utility issues one RESTORE command for each group of objects that is associated with the concurrent copy data set. If you do not use the CURRENTCOPYONLY keyword, RECOVER issues one RESTORE command for each object.

Optimizing the LOGAPPLY phase

The time that is required to recover a table space depends on the time that is required to read and apply log data. You can take several steps to optimize the process.

If possible, DB2 reads the required log records from the active log to provide the best performance.

Any log records that are not found in the active logs are read from the archive log data sets, which are dynamically allocated to satisfy the requests. The type of

storage that is used for archive log data sets is a significant factor in the performance. Consider the following actions to improve performance:

- RECOVER a list of objects in one utility statement to take only a single pass of the log.
- Keep archive logs on disk to provide the best possible performance.
- Control archive logs data sets by using DFSMShsm to provide the next best performance. DB2 optimizes recall of the data sets. After the data set is recalled, DB2 reads it from disk.
- If the archive log must be read from tape, DB2 optimizes access by means of ready-to-process and look-ahead mount requests. DB2 also permits delaying the deallocation of a tape drive if subsequent RECOVER jobs require the same archive log tape. Those methods are described in more detail in the subsequent paragraphs.

The BSDS contains information about which log data sets to use and where they reside. You must keep the BSDS information current. If the archive log data sets are cataloged, the ICF catalog indicates where to allocate the required data set.

DFSMShsm data sets: The recall of the first DFSMShsm archive log data set starts automatically when the LOGAPPLY phase starts. When the recall is complete and the first log record is read, the recall for the next archive log data set starts. This process is known as *look-ahead* recalling. Its purpose is to recall the next data set while it reads the preceding one.

When a recall is complete, the data set is available to all RECOVER jobs that require it. Reading proceeds in parallel.

Non-DFSMShsm tape data sets: DB2 reports on the console all tape volumes that are required for the entire job. The report distinguishes two types of volumes:

- Any volume that is **not** marked with an asterisk (*) is **required** for the for the job to complete. Obtain these volumes from the tape library as soon as possible.
- Any volume that **is** marked with an asterisk (*) contains data that is also contained in one of the active log data sets. The volume might or might not be required.

As tapes are mounted and read, DB2 makes two types of mount requests:

- *Ready-to-process*: The current job needs this tape immediately. As soon as the tape is loaded, DB2 allocates and opens it.
- *Look-ahead*: This is the next tape volume that is required by the current job. Responding to this request enables DB2 to allocate and open the data set before it is needed, thus reducing overall elapsed time for the job.

You can dynamically change the maximum number of input tape units that are used to read the archive log by specifying the COUNT option of the SET ARCHIVE command. For example, use the following command to assign 10 tape units to your DB2 subsystem:

```
-SET ARCHIVE COUNT (10)
```

The DISPLAY ARCHIVE READ command shows the currently mounted tape volumes and their statuses.

Delayed deallocation: DB2 can delay deallocating the tape units used to read the archive logs. This is useful when several RECOVER utility statements run in

parallel. By delaying deallocation, DB2 can re-read the same volume on the same tape unit for different RECOVER jobs, without taking time to allocate it again.

You can dynamically change the amount of time that DB2 delays deallocation by using the TIME option of the SET ARCHIVE command. For example, to specify a 60 minute delay, issue the following command:

```
-SET ARCHIVE TIME(60)
```

In a data sharing environment, you might want to specify zero (0) to avoid having one member hold onto a data set that another member needs for recovery.

Performance summary:

1. Achieve the best performance by allocating archive logs on disk.
2. Consider staging cataloged tape data sets to disk before allocation by the log read process.
3. If the data sets are read from tape, set both the COUNT and the TIME values to the maximum allowable values within the system constraints.

Recovering image copies in a JES3 environment

Ensure that sufficient units are available to mount the required image copies. In a JES3 environment, if the number of image copies that need to be restored exceeds the number of available online and offline units, and the RECOVER job successfully allocates all available units, the job waits for more units to become available.

Resetting RECOVER-pending or REBUILD pending status

Several possible operations on a table space can place the table space in the RECOVER-pending status and the index space in REBUILD-pending status. You can turn off the status in several ways, as follows:

- Recover the table space, index space, or partition.
- Use REBUILD INDEX to rebuild the index space from existing data.
- Use the LOAD utility, with the REPLACE option, on the table space or partition.
- Use the REPAIR utility, with the NORCVRPEND option, on the table space, index space, or partition. Be aware that the REPAIR utility does not fix the data inconsistency in the table space or index.
- To rebuild indexes, run REORG TABLESPACE SORTDATA for table spaces and indexes.

Allocating incremental image copies

RECOVER attempts to dynamically allocate all required incremental image copy data sets. If any of the incremental image copies are missing, RECOVER performs the following actions:

- Identifies the first incremental image copy that is missing
- Uses the incremental image copies up to the missing incremental image copy
- Doesn't use the remaining incremental image copy data sets
- Applies additional log records to compensate for any incremental image copies that were not used

For example, if the incremental image copies are on tape and an adequate number of tape drives are not available, RECOVER does not use the remaining incremental image copy data sets.

Performing fallback recovery

If the RECOVER utility cannot use the latest primary copy data set as a starting point for recovery, it attempts to use the backup copy data set, if one is available. If neither image copy is usable, RECOVER attempts to fall back to a previous recovery point. If the previous recovery point is a full image copy, the RECOVER utility uses the full image copy, any incremental image copies, and the log to recover. If a previous REORG LOG YES or LOAD REPLACE LOG YES was done, RECOVER attempts to recover from the log and applies any changes that occurred between the two image copies. If good full image copies are not available, and no previous REORG LOG YES or LOAD REPLACE LOG YES jobs were run, the RECOVER utility terminates.

If one of the following actions occurs, the index remains untouched, and utility processing terminates with return code 8:

- RECOVER processes an index for which no full copy exists.
- The copy cannot be used because of utility activity that occurred on the index or on its underlying table space,

For more information, see 125.

If you always make multiple image copies, RECOVER should seldom fall back to an earlier point. Instead, RECOVER relies on the backup copy data set if the primary copy data set is unusable.

In a JES3 environment, you can do a fallback recovery by issuing a JES3 `cancel,s` command at the time the allocation mount message is issued. This action might be necessary if a volume is not available or if the given volume is not desired.

RECOVER does not perform parallel processing for objects that are in backup or fallback recovery. Instead, the utility performs non-parallel image copy allocation processing of the objects. RECOVER defers the processing of objects that require backup or fallback processing until all other objects are recovered, at which time the utility processes the objects one at a time.

Retaining tape mounts

DB2 can retain tape mounts for you. For image copies on one or more tape volumes, you do not need to code JCL statements to retain the tape mounts. Instead, use the PARALLEL and TAPEUNITS keywords to control the allocation of tape devices for the job. In some cases, RECOVER cannot retain all tape mounts so tapes might be deallocated, even if you specify the PARALLEL and TAPEUNITS keywords.

Avoiding damaged media

When a media error is detected, DB2 prints a message that indicates the extent of the damage. If an entire volume is bad and storage groups are being used, you must remove the bad volume first; otherwise, the RECOVER utility might re-access the damaged media. You must follow these steps:

1. Use ALTER STOGROUP to remove the bad volume and add another volume.
2. Execute the RECOVER utility for all objects on that volume.

If the RECOVER utility cannot complete because of severe errors that are caused by the damaged media, you might need to use Access Method Services (IDCAMS) with the NOSCRATCH option to delete the cluster for the table space or index. If the table space or index is defined by using STOGROUP, the RECOVER utility automatically redefines the cluster. For user-defined table spaces or indexes, you must redefine the cluster before invoking the RECOVER utility.

Recovering table spaces and index spaces with mixed volume IDs

You cannot run RECOVER on a table space or index space on which mixed specific and non-specific volume IDs were defined with CREATE STOGROUP or ALTER STOGROUP.

Terminating or restarting RECOVER

This section contains information about how to terminate and restart RECOVER.

Terminating RECOVER

Terminating a RECOVER job with the TERM UTILITY command leaves the table space that is being recovered in RECOVER-pending status, and the index space that is being recovered in the REBUILD-pending status. If you recover a table space to a previous point in time, its indexes are left in the REBUILD-pending status. The data or index is unavailable until the object is successfully recovered or rebuilt.

Restarting RECOVER

You can restart RECOVER from the last commit point (RESTART(CURRENT)) or the beginning of the phase (RESTART(PHASE)). By default, DB2 uses RESTART(CURRENT).

If you attempt to recover multiple objects by using a single RECOVER statement and the utility fails in:

- The RESTORE phase: All objects in the process of being restored are placed in the RECOVER-pending or REBUILD-pending status. The status of the remaining objects is unchanged.
- The LOGAPPLY phase: All objects that are specified in the RECOVER statement are placed in the RECOVER-pending or REBUILD-pending status.

In both cases, you must identify and fix the causes of the failure before performing a current restart.

For instructions on restarting a utility job, see “Restarting an online utility” on page 43.

Concurrency and compatibility for RECOVER

DB2 treats individual data and index partitions as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible. However, if a nonpartitioned secondary index exists on a partitioned table space, utilities that operate on different partitions of a table space can be incompatible because of contention on the nonpartitioned secondary index.

Table 61 shows which claim classes RECOVER claims and drains and any restrictive state that the utility sets on the target object.

Table 61. Claim classes of RECOVER operations.

Target	RECOVER (no option)	RECOVER TORBA or TOCOPY	RECOVER PART TORBA or TOCOPY	RECOVER ERROR-RANGE
Table space or partition	DA/UTUT	DA/UTUT	DA/UTUT	DA/UTUT CW/UTRW[1]

Table 61. Claim classes of RECOVER operations. (continued)

Target	RECOVER (no option)	RECOVER TORBA or TOCOPY	RECOVER PART TORBA or TOCOPY	RECOVER ERROR-RANGE
Partitioning index, data-partitioned secondary index, or physical partition	DA/UTUT	DA/UTUT	DA/UTUT	DA/UTUT CW/UTRW[1]
Nonpartitioned secondary index	DA/UTUT	DA/UTUT	DA/UTUT	DA/UTUT CW/UTRW[1]
RI dependents	none	CHKP (YES)	CHKP (YES)	none

Legend:

- CHKP (YES): Concurrently running applications enter CHECK-pending after commit
- CW: Claim the write claim class
- DA: Drain all claim classes, no concurrent SQL access
- DR: Drain the repeatable read class, no concurrent access for SQL repeatable readers
- RI: Referential integrity
- UTRW: Utility restrictive state, read-write access allowed
- UTUT: Utility restrictive state, exclusive control
- none: Object is not affected by this utility

Notes:

1. During the UTILINIT phase, the claim and restrictive states change from DA/UTUT to CW/UTRW.

RECOVER does not set a utility restrictive state if the target object is DSNDB01.SYSUTILX.

Table 62 shows which utilities can run concurrently with RECOVER on the same target object. The target object can be a table space, an index space, or a partition of a table space or index space. If compatibility depends on particular options of a utility, that information is also documented in the table.

Table 62. Compatibility of RECOVER with other utilities

Action	Compatible with RECOVER (no option)?	Compatible with RECOVER TOCOPY or TORBA?	Compatible with RECOVER ERROR-RANGE?
CHECK DATA	No	No	No
CHECK INDEX	No	No	No
CHECK LOB	No	No	No
COPY INDEXSPACE	No	No	No
COPY TABLESPACE	No	No	No
DIAGNOSE	Yes	Yes	Yes
LOAD	No	No	No
MERGECOPY	No	No	No
MODIFY	No	No	No
QUIESCE	No	No	No

Table 62. Compatibility of RECOVER with other utilities (continued)

Action	Compatible with RECOVER (no option)?	Compatible with RECOVER TOCOPY or TORBA?	Compatible with RECOVER ERROR-RANGE?
REBUILD INDEX	No	No	No
REORG INDEX	Yes	No	Yes
REORG TABLESPACE	No	No	No
REPAIR LOCATE INDEX	Yes	No	Yes
REPAIR LOCATE TABLESPACE	No	No	No
REPORT	Yes	Yes	Yes
RUNSTATS INDEX	No	No	No
RUNSTATS TABLESPACE	No	No	No
STOSPACE	Yes	Yes	Yes
UNLOAD	No	No	No

To run on DSNDB01.SYSUTILX, RECOVER must be the only utility in the job step and the only utility running in the DB2 subsystem.

RECOVER on any catalog or directory table space is an exclusive job; such a job can interrupt another job between job steps, possibly causing the interrupted job to time out.

Effects of running RECOVER

When you run RECOVER without the REUSE option and the data set that contains that data is DB2–managed, DB2 deletes this data set before the RECOVER and redefines a new data set with a control interval that matches the page size.

Sample RECOVER control statements

Example 1: Recovering a table space. The following control statement specifies that the RECOVER utility is to recover table space DSN8D81A.DSN8S81D to the current point in time.

```
RECOVER TABLESPACE DSN8D81A.DSN8S81D
```

Example 2: Recovering a table space partition. The following control statement specifies that the RECOVER utility is to recover the second partition of table space DSN8D81A.DSN8S81D. The partition number is indicated by the DSNUM option.

```
RECOVER TABLESPACE DSN8D81A.DSN8S81D DSNUM 2
```

Example 3: Recovering a table space partition to the last image copy that was taken. The following control statement specifies that the RECOVER utility is to recover the first partition of table space DSN8D81A.DSN8S81D to the last image copy that was taken. If the last image copy that was taken is a full image copy, this full image copy is restored. If the last image copy that was taken is an incremental image copy, the most recent full image copy, along with any incremental image copies, are restored.

```
RECOVER TABLESPACE DSN8D81A.DSN8S81D DSNUM 1 TOLASTCOPY
```

Example 4: Recovering table spaces to a point in time. The following control statement specifies that the RECOVER utility is to recover the second partition of table space DSN8D81A.DSN8S81E and all of table space DSN8D81A.DSN8S81D to the indicated quiesce point (LRSN X'00000551BE7D'). The quiesce point is indicated by the TOLOGPOINT option. Note that the value for this option can be either an LRSN or an RBA.

```
RECOVER TABLESPACE DSN8D81A.DSN8S81E DSNUM 2
        TABLESPACE DSN8D81A.DSN8S81D
        TOLOGPOINT X'00000551BE7D'
```

Example 5: Recovering an index to the last full image copy that was taken without deleting and redefining the data sets. The following control statement specifies that the RECOVER utility is to recover index ADMF001.IADH082P to the last full image copy. The REUSE option specifies that DB2 is to logically reset and reuse DB2-managed data sets without deleting and redefining them.

```
RECOVER INDEX  ADMF001.IADH082P REUSE TOLASTFULLCOPY
```

Example 6: Recovering from concurrent copies. The RECOVER utility control statement in Figure 74 specifies that the utility is to recover all of the objects that are included in the RCVR4_LIST. This list is defined by the preceding LISTDEF utility control statement. Because the most recent primary copy for all of these objects is a concurrent copy, the CURRENTCOPYONLY option is used in the RECOVER statement to improve the performance of restoring these concurrent copies. The LOCALSITE option indicates that RECOVER is to use image copies at the local site.

```
//STEP1    EXEC DSNUPROC,UID='JUOLU210.RCVR4',
//         UTPROC='',
//         SYSTEM='SSTR'
//UTPRINT  DD  SYSOUT=*
//SYSUT1   DD DSN=JUOLU210.RCVR4.STEP1.SYSUT1,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT  DD DSN=JUOLU210.RCVR4.STEP1.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *

   LISTDEF RCVR4_LIST
     INCLUDE TABLESPACES TABLESPACE DBOL1002.TSOL1002
     INCLUDE TABLESPACES TABLESPACE DBOL1003.TPOL1003 PARTLEVEL 3
     INCLUDE TABLESPACES TABLESPACE DBOL1003.TPOL1003 PARTLEVEL 6
     INCLUDE TABLESPACES TABLESPACE DBOL1003.TPOL1004 PARTLEVEL 5
     INCLUDE TABLESPACES TABLESPACE DBOL1003.TPOL1004 PARTLEVEL 9
     INCLUDE INDEXSPACES INDEXSPACE DBOL1003.IPOL1051 PARTLEVEL 22
     INCLUDE INDEXSPACES INDEXSPACE DBOL1003.IPOL1061 PARTLEVEL 10
     INCLUDE INDEXSPACES INDEXSPACE DBOL1003.IXOL1062

   RECOVER LIST RCVR4_LIST
           LOCALSITE
           CURRENTCOPYONLY
 /*
```

Figure 74. Example RECOVER control statement with the CURRENTCOPYONLY option

Example 7: Recovering a list of objects on different tape devices in parallel. The control statement in Figure 75 on page 388 specifies that the RECOVER utility is to recover the list of table spaces. Full image copies and incremental image copies of the eight table spaces are stacked on four different tape volumes. The utility sorts the list of objects and, if possible, recovers two objects at a time in parallel. This number of objects is specified by the PARALLEL option. The TAPEUNITS option

specifies that up to four tape drives are to be dynamically allocated.

```
//RECOVER   EXEC DSNUPROC,SYSTEM='DSN'
//SYSIN    DD *
RECOVER PARALLEL(2) TAPEUNITS(4)
TABLESPACE DB1.TS8
TABLESPACE DB1.TS7
TABLESPACE DB1.TS6
TABLESPACE DB1.TS5
TABLESPACE DB1.TS4
TABLESPACE DB1.TS3
TABLESPACE DB1.TS2
TABLESPACE DB1.TS1
```

Figure 75. Example RECOVER control statement for a list of objects on tape

Example 8: Recovering a list of objects to a point in time. The following RECOVER control statement specifies that the RECOVER utility is to recover the specified list of objects to a common quiesce point (LRSN X'00000551BE7D'). The LISTDEF control statement defines which objects are to be included in the list. These objects are logically consistent after successful completion of this RECOVER job. The PARALLEL option indicates that RECOVER is to restore four objects at a time in parallel. If any of the image copies are on tape (either stacked or not stacked), RECOVER determines the number of tape drives to use to optimize the process.

```
LISTDEF RCVRLIST INCLUDE TABLESPACE DSN8D81A.DSN8S81D
         INCLUDE INDEX DSN8810.XDEPT1
         INCLUDE INDEX DSN8810.XDEPT2
         INCLUDE INDEX DSN8810.XDEPT3
         INCLUDE TABLESPACE DSN8D81A.DSN8S81E
         INCLUDE INDEX DSN8810.XEMP1
         INCLUDE INDEX DSN8810.XEMP2
RECOVER LIST RCVRLIST TOLOGPOINT X'00000551BE7D' PARALLEL(4)
```

Chapter 24. REORG INDEX

The online REORG INDEX utility reorganizes an index space to improve access performance and reclaim fragmented space. You can specify the degree of access to your data during reorganization, and you can collect inline statistics by using the STATISTICS keyword.

You can determine when to run REORG INDEX by using the LEAFDISTLIMIT catalog query option. If you specify the REPORTONLY option, REORG INDEX produces a report that indicates whether if a REORG is recommended; in this case, a REORG is not performed. These options are not available for indexes on the directory.

For a diagram of REORG INDEX syntax and a description of available options, see "Syntax and options of the REORG INDEX control statement" on page 390. For detailed guidance on running this utility, see "Instructions for running REORG INDEX" on page 402.

Output: The following list summarizes REORG INDEX output:

REORG specified	Results
REORG INDEX	Reorganizes the entire index (all parts if partitioning).
REORG INDEX PART *n*	Reorganizes PART *n* of a partitioning index or of a data-partitioned secondary index

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:
- REORG privilege for the database
- DBADM or DBCTRL authority for the database
- SYSCTRL authority
- SYSADM authority

To execute this utility on an index space in the catalog or directory, you must use a privilege set that includes one of the following authorities:
- REORG privilege for the DSNDB06 (catalog) database
- DBADM or DBCTRL authority for the DSNDB06 (catalog) database
- Installation SYSOPR authority
- SYSCTRL authority
- SYSADM or Installation SYSADM authority

While trying to reorganize an index space in the catalog or directory, a user with authority other than installation SYSADM or installation SYSOPR might receive the following message:

```
DSNT500I "resource unavailable"
```

This message is issued when the DSNDB06.SYSDBAUT or DSNDB06.SYSUSER catalog table space or one of the indexes is unavailable. If this problem occurs, run the REORG INDEX utility again, using an authorization ID with the installation SYSADM or installation SYSOPR authority.

An ID with installation SYSOPR authority can also execute REORG INDEX, but only on an index in the DSNDB06 database.

To run REORG INDEX STATISTICS REPORT YES, you must use a privilege set that includes STATS privilege for the database and the SELECT privilege on the catalog tables and tables for which statistics are to be gathered. REORG INDEX STATISTICS REPORT ALL does not report values from tables that the user is not authorized to see.

Execution phases of REORG INDEX: The REORG INDEX utility operates in these phases:

Phase	Description
UTILINIT	Performs initialization and setup
UNLOAD	Unloads index space and writes keys to a sequential data set.
BUILD	Builds indexes. Updates index statistics.
LOG	Processes log iteratively. Used only if you specify SHRLEVEL CHANGE.
SWITCH	Switches access between original and new copy of index space or partition. Used only if you specify SHRLEVEL REFERENCE or CHANGE.
UTILTERM	Performs cleanup. For DB2-managed data sets and either SHRLEVEL CHANGE or SHRLEVEL REFERENCE, the utility deletes the original copy of the table space or index space.

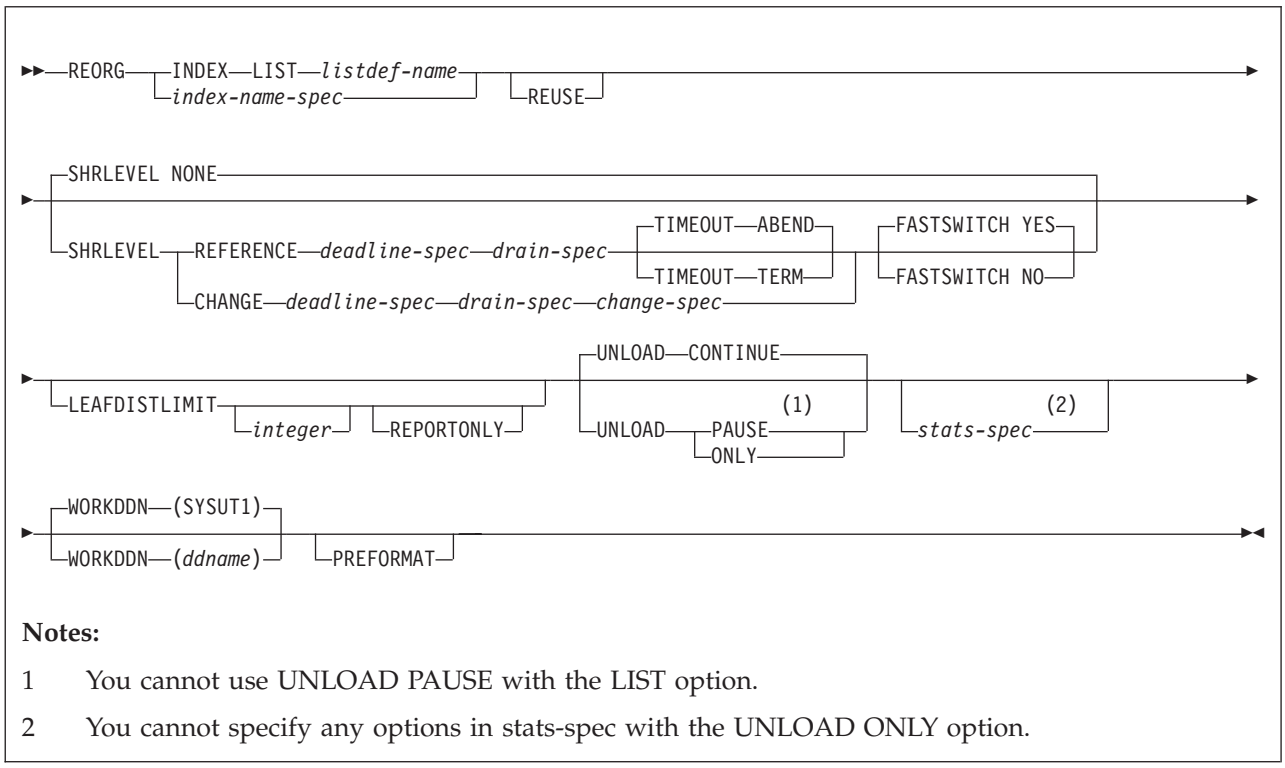
The following topics provide additional information:

- "Syntax and options of the REORG INDEX control statement"
- "Instructions for running REORG INDEX" on page 402
- "Concurrency and compatibility for REORG INDEX" on page 413
- "Reviewing REORG INDEX output" on page 414
- "The effect of REORG INDEX on index version numbers" on page 415
- "Sample REORG INDEX control statements" on page 415

Syntax and options of the REORG INDEX control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram

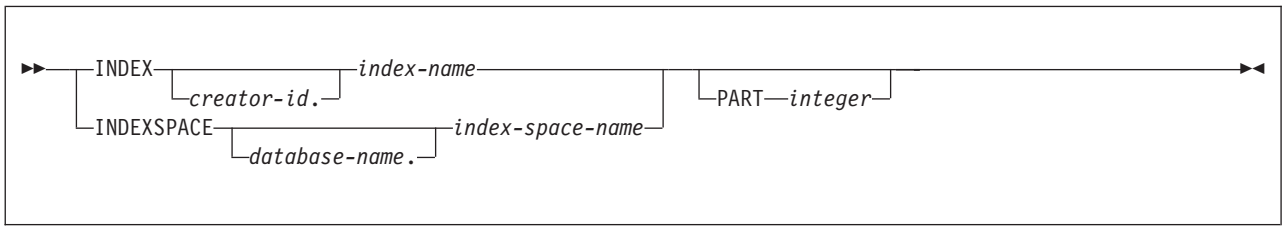


Notes:

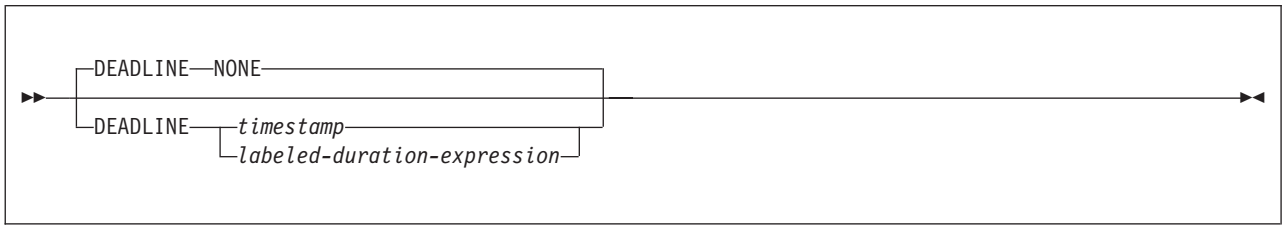
1 You cannot use UNLOAD PAUSE with the LIST option.

2 You cannot specify any options in stats-spec with the UNLOAD ONLY option.

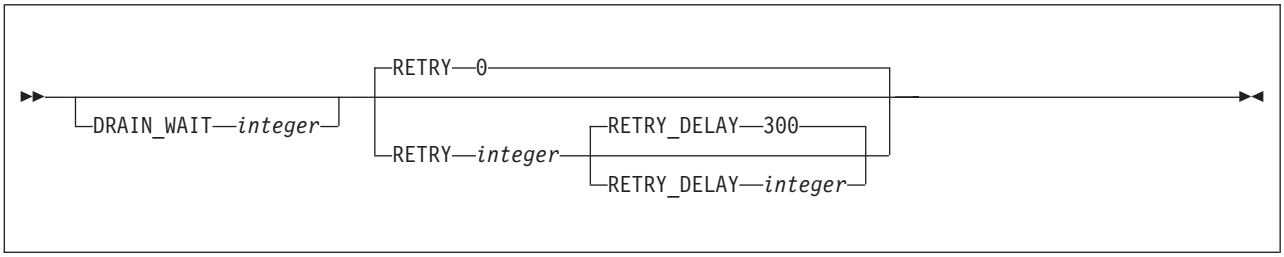
index-name-spec:



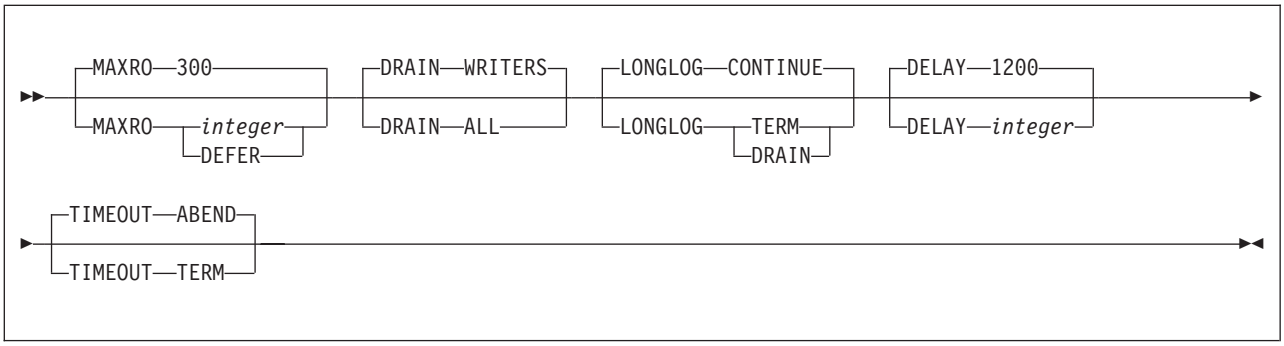
deadline-spec:



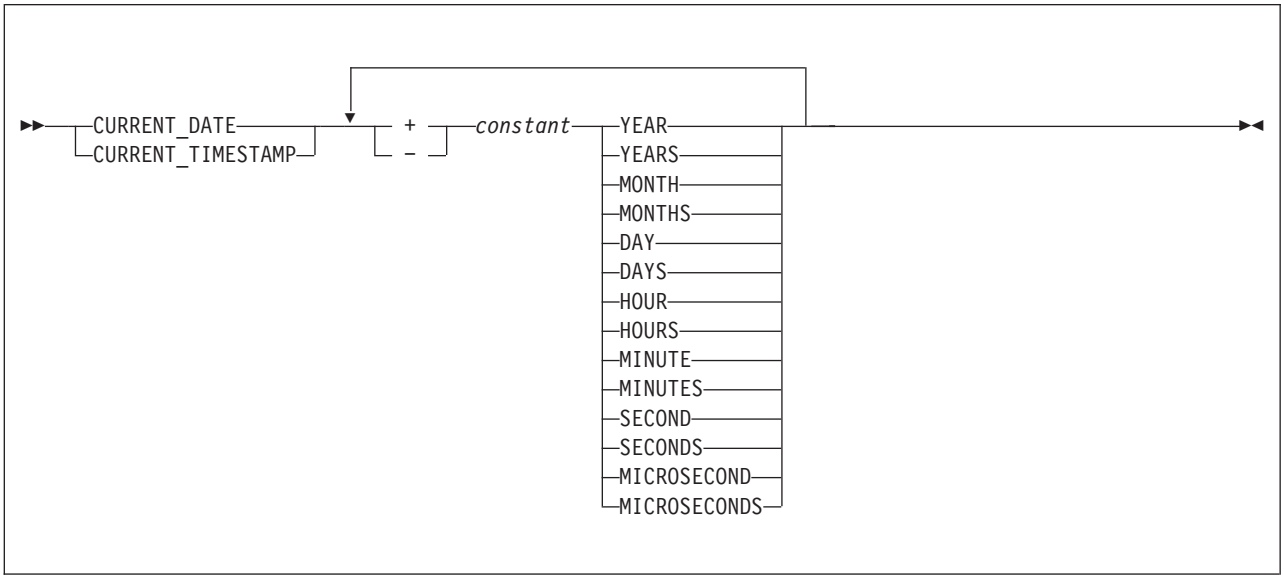
drain-spec:



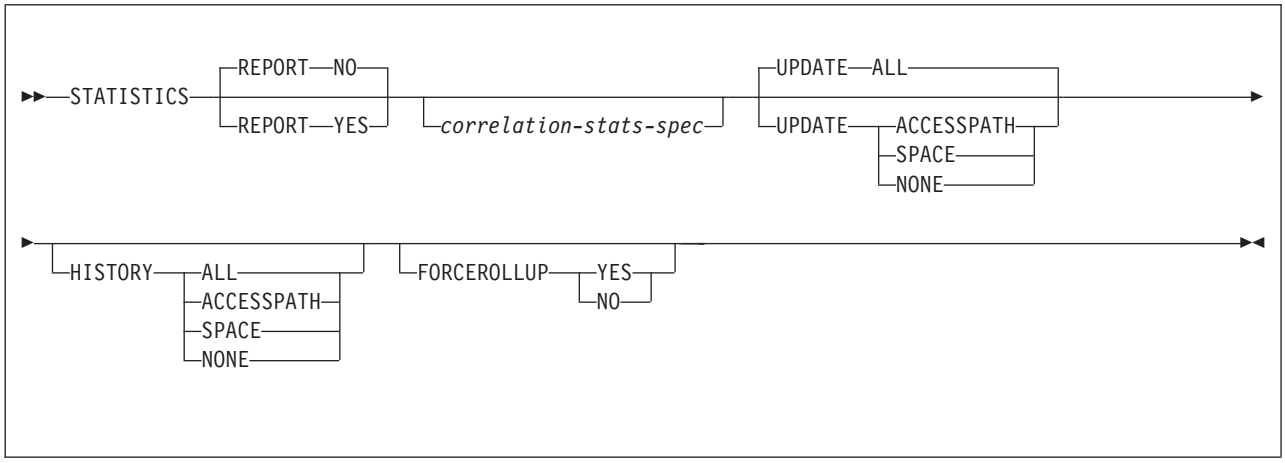
change-spec:



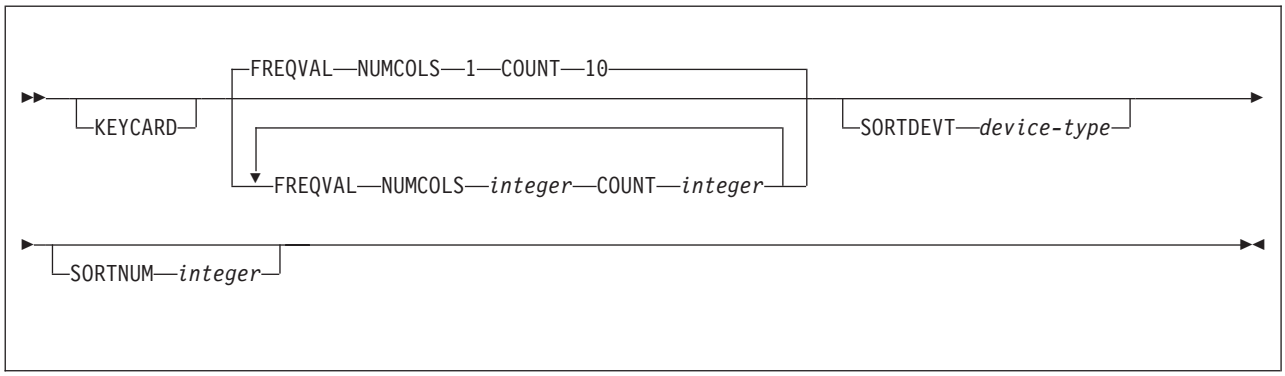
labeled-duration-expression:



stats-spec:



correlation-stats-spec:



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

INDEX *creator-id.index-name*
Specifies an index that is to be reorganized.

creator-id. specifies the creator of the index and is optional. If you omit the qualifier creator id, DB2 uses the user identifier for the utility job. *index-name* is the qualified name of the index to copy. For an index, you can specify either an index name or an index space name. Enclose the index name in quotation marks if the name contains a blank.

INDEXSPACE *database-name.index-space-name*
Specifies the qualified name of the index space that is obtained from the SYSIBM.SYSINDEXES table.

database-name specifies the name of the database that is associated with the index and is optional. The **default** is **DSNDB04**.

index-space-name specifies the qualified name of the index space that is to be reorganized; the name is obtained from the SYSIBM.SYSINDEXES table.

LIST *listdef-name*
Specifies the name of a previously defined LISTDEF list name. The INDEX

keyword is required to differentiate this REORG INDEX LIST from REORG TABLESPACE LIST. The utility allows one LIST keyword for each control statement of REORG INDEX. The list must not contain any table spaces. REORG INDEX is invoked once for each item in the list. For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

Do not specify STATISTICS INDEX *index-name* with REORG INDEX LIST. If you want to collect inline statistics for a list of indexes, just specify STATISTICS.

You cannot specify DSNUM and PART with LIST on any utility.

REUSE
When used with SHRLEVEL NONE, specifies that REORG is to logically reset and reuse DB2-managed data sets without deleting and redefining them. If you do not specify REUSE and SHRLEVEL NONE, DB2 deletes and redefines DB2-managed data sets to reset them.

If a data set has multiple extents and you use the REUSE parameter, the extents are not released.

If you specify SHRLEVEL REFERENCE or CHANGE with REUSE, REUSE does not apply

PART *integer*
Identifies a partition that is to be reorganized. You can reorganize a single partition of a partitioning index. You cannot specify PART with LIST. *integer* must be in the range from 1 to the number of partitions that are defined for the partitioning index. The maximum is 4096.

integer designates a single partition.

If you omit the PART keyword, the entire index is reorganized.

SHRLEVEL
Specifies the method for performing the reorganization. The parameter following SHRLEVEL indicates the type of access that is to be allowed during the RELOAD phase of REORG.

NONE
Specifies that reorganization is to operate by unloading from the area that is being reorganized (while applications can read but cannot write to the area), building into that area (while applications have no access), and then allowing read-write access again. The **default** is **NONE**.

If you specify NONE (explicitly or by default), you cannot specify the following parameters:
- MAXRO
- LONGLOG
- DELAY
- DEADLINE
- DRAIN_WAIT
- RETRY
- RETRY_DELAY

REFERENCE
Specifies that reorganization is to operate as follows:
- Unload from the area that is being reorganized while applications can read but cannot write to the area.
- Build into a shadow copy of that area while applications can read but cannot write to the original copy.

- Switch the future access of the applications from the original copy to the shadow copy by exchanging the names of the data sets, and then allowing read-write access again.

To determine which data sets are required when you execute REORG SHRLEVEL REFERENCE, see "Data sets that REORG INDEX uses" on page 403.

If you specify REFERENCE, you cannot specify the following parameters:

- UNLOAD (Reorganization with REFERENCE always performs UNLOAD CONTINUE.)
- MAXRO
- LONGLOG
- DELAY

CHANGE

Specifies that reorganization is to operate as follows:

- Unload from the area that is being reorganized while applications can read and write to the area.
- Build into a shadow copy of that area while applications can read and write to the original copy.
- Apply the log of the original copy to the shadow copy while applications can read and usually write to the original copy.
- Switch the future access of the applications from the original copy to the shadow copy by exchanging the names of the data sets, and then allowing read-write access again.

To determine which data sets are required when you execute REORG SHRLEVEL CHANGE, see "Data sets that REORG INDEX uses" on page 403.

If you specify CHANGE, you cannot specify the UNLOAD parameter. Reorganization with CHANGE always performs UNLOAD CONTINUE.

DEADLINE

Specifies the deadline for the SWITCH phase to begin. If DB2 estimates that the SWITCH phase does not begin by the deadline, DB2 issues the messages that the DISPLAY UTILITY command issues and then terminates reorganization.

NONE

Specifies that no deadline exists by which the switch phase of log processing must begin. The **default** is **NONE.**

timestamp

Specifies the deadline for the switch phase of log processing to begin. This deadline must not have already occurred when REORG is executed.

labeled-duration-expression

Calculates the deadline for the switch phase of log processing to begin. The calculation is based on either CURRENT TIMESTAMP or CURRENT DATE. You can add or subtract one or more *constant* values to specify the deadline. This deadline must not have already occurred when REORG is executed. CURRENT TIMESTAMP and CURRENT DATE are evaluated once, when the REORG statement is first processed. If a list of objects is specified, the same value will be in effect for all objects in the list.

CURRENT_DATE
Specifies that the deadline is to be calculated based on the CURRENT DATE.

CURRENT_TIMESTAMP
Specifies that the deadline is to be calculated based on the CURRENT TIMESTAMP.

constant
Indicates a unit of time and is followed by one of the seven duration keywords: YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS, or MICROSECONDS. The singular form of these words is also acceptable: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, MICROSECOND.

If REORG SHRLEVEL REFERENCE or SHRLEVEL CHANGE terminates because of a DEADLINE specification, DB2 issues message DSNU374I with reason code 2 but does not set a restrictive status.

DRAIN_WAIT *integer*
Specifies the number of seconds that the utility waits when draining for SQL statements (inserts, updates, deletes, and selects). The specified time is the aggregate time for all partitions of the index that is to be reorganized. This value overrides the values specified by IRLMRWT and UTIMOUT, for these SQL statements only. For operations like COMMANDS, the IRLMRWT and UTIMOUT values are used. If the keyword is omitted or a value of 0 is specified, the utility uses the IRLMRWT and UTIMOUT values for regular draining. Valid values for *integer* are from 0 to 1800.

RETRY *integer*
Specifies the maximum number of retries that REORG is to attempt. Valid values for *integer* are from 0 to 255. If the keyword is omitted, the utility does not attempt a retry.

Specifying RETRY can lead to increased processing costs and can result in multiple or extended periods of read-only access.

RETRY_DELAY *integer*
Specifies the minimum duration, in seconds, between retries. Valid values for *integer* are from 1 to 1800. The **default** is **300** seconds.

MAXRO *integer*
Specifies the maximum amount of time for the last iteration of log processing. During that iteration, applications have read-only access.

The actual execution time of the last iteration might exceed the specified MAXRO value.

The ALTER UTILITY command can change the value of MAXRO.

integer *integer* is the number of seconds. Specifying a small positive value reduces the length of the period of read-only access, but it might increase the elapsed time for REORG to complete. If you specify a huge positive value, the second iteration of log processing is probably the last iteration. The **default** is **300** seconds.

DEFER
Specifies that the iterations of log processing with read-write access can continue indefinitely. REORG never begins the final iteration with read-only access, unless you change the MAXRO value by using the ALTER UTILITY command.

If you specify DEFER, you should also specify LONGLOG CONTINUE.

If you specify DEFER, and DB2 determines that the actual time for an iteration and the estimated time for the next iteration are both less than 5 seconds, DB2 adds a 5-second pause to the next iteration. This pause reduces consumption of processor time. The first time this situation occurs for a given execution of REORG, DB2 sends message DSNU362I to the console. The message states that the number of log records that must be processed is small and that the pause occurs. To change the MAXRO value and thus cause REORG to finish, execute the ALTER UTILITY command. DB2 adds the pause whenever the situation occurs; however, DB2 sends the message only if 30 minutes have elapsed since the last message was sent for a given execution of REORG.

DRAIN
Specifies drain behavior at the end of the log phase after the MAXRO threshold is reached and when the last iteration of the log is to be applied.

WRITERS
Specifies the current default action, in which DB2 drains only the writers during the log phase after the MAXRO threshold is reached and subsequently issues DRAIN ALL on entering the switch phase.

ALL Specifies that DB2 is to drain all readers and writers during the log phase, after the MAXRO threshold is reached.

Consider specifying DRAIN ALL if the following conditions are both true:

- SQL update activity is high during the log phase.
- The default behavior results in a large number of -911 SQL error messages.

LONGLOG
Specifies the action that DB2 is to perform, after sending a message to the console, if the number of records that the next iteration of log process is to process is not sufficiently lower than the number that the previous iterations processed. This situation means that REORG INDEX is not reading the application log quickly enough to keep pace with the writing of the application log.

CONTINUE
Specifies that until the time on the JOB statement expires, DB2 is to continue performing reorganization, including iterations of log processing, if the estimated time to perform an iteration exceeds the time that is specified with MAXRO.

A value of DEFER for MAXRO and a value of CONTINUE for LONGLOG together mean that REORG INDEX is to continue allowing access to the original copy of the area that is being reorganized and does not switch to the shadow copy. The user can execute the ALTER UTILITY command with a large value for MAXRO when the switching is desired.

The **default** is **CONTINUE.**

TERM Specifies that DB2 is to terminate reorganization after the delay specified by the DELAY parameter.

DRAIN
Specifies that DB2 is to drain the write claim class after the delay that is specified by the DELAY parameter. This action forces the final iteration of log processing to occur.

DELAY *integer*
Specifies the minimum interval between the time that REORG sends the LONGLOG message to the console and the time REORG that performs the action that is specified by the LONGLOG parameter.

integer is the number of seconds. The **default** is **1200**.

TIMEOUT
Specifies the action that is to be taken if the REORG INDEX utility gets a time-out condition while trying to drain objects in either the log or switch phases.

ABEND
Indicates that if a time-out condition occurs, DB2 is to leave the objects in a UTRO or UTUT state.

TERM
Indicates that DB2 is to behave as follows if you specify the TERM option and a time out condition occurs:

1. DB2 issues an implicit TERM UTILITY command, causing the utility to end with a return code 8.
2. DB2 issues the DSNU590I and DSNU170I messages.
3. DB2 leaves the objects in a RW state.

FASTSWITCH
Specifies which switch methodology is to be used.

YES
Specifies that the fifth-level qualifier in the data set name is to alternate between I0001 and J0001. This option is not allowed for the catalog (DSNDB06) or directory (DSNDB01). The **default** is FASTSWITCH **YES**.

NO
Specifies that the SWITCH phase is to use IDCAMS RENAME.

LEAFDISTLIMIT *integer*
Specifies that the value for *integer* is to be compared to the LEAFDIST value for the specified partitions of the specified index in SYSIBM.SYSINDEXPART. If any LEAFDIST value exceeds the specified LEAFDISTLIMIT value, REORG is performed or, if you specify REPORTONLY, recommended.

The **default** value is **200**.

REPORTONLY
Specifies that REORG is only to be recommended, not performed. REORG produces a report with one of the following return codes:
1 No limit met; no REORG performed or recommended.
2 REORG performed or recommended.

UNLOAD
Specifies whether the utility job is to continue processing or terminate after the data is unloaded.

CONTINUE
Specifies that, after the data has been unloaded, the utility is to continue processing. The **default** is **CONTINUE**.

PAUSE
Specifies that, after the data has been unloaded, processing is to end. The utility stops and the RELOAD status is stored in SYSIBM.SYSUTIL so that processing can be restarted with RELOAD RESTART(PHASE).

This option is useful if you want to redefine data sets during reorganization. For example, with a user-defined data set, you can:
- Run REORG with the UNLOAD PAUSE option.
- Redefine the data set using Access Method Services.
- Restart REORG by resubmitting the previous job and specifying RESTART(PHASE).

If no records are unloaded during an UNLOAD PAUSE, when REORG is restarted, the RELOAD and BUILD phases are bypassed.

You cannot use UNLOAD PAUSE if you specify the LIST option.

ONLY Specifies that, after the data has been unloaded, the utility job ends and the status in SYSIBM.SYSUTIL that corresponds to this utility ID is removed.

STATISTICS
Specifies that statistics for the index are to be collected; the statistics are either reported or stored in the DB2 catalog. You cannot collect inline statistics for indexes on the catalog and directory tables.

Restriction: If you specify STATISTICS for encrypted data, DB2 might not
provide useful information on this data. If the utility is terminated with the
-TERM UTIL command after the STATISTICS have been updated in the
catalog, the statistics are not rolled back. A subsequent RUNSTATS utility may
be needed.

REPORT
Indicates whether a set of messages is to be generated to report the collected statistics.

NO
Indicates that the set of messages is not to be sent as output to SYSPRINT.

The **default** is **NO**.

YES
Indicates that the set of messages is to be sent as output to SYSPRINT. The generated messages are dependent on the combination of keywords (such as TABLESPACE, INDEX, TABLE, and COLUMN) that are specified with the RUNSTATS utility. However, these messages are **not** dependent on the specification of the UPDATE option. REPORT YES always generates a report of SPACE and ACCESSPATH statistics.

KEYCARD
Indicates that all of the distinct values in all of the 1 to *n* key column combinations for the specified indexes are to be collected. *n* is the number of columns in the index.

FREQVAL
Specifies that frequent value statistics are to be collected. If you specify FREQVAL, you must also specify NUMCOLS and COUNT.

NUMCOLS
Indicates the number of key columns to concatenate together when you collect frequent values from the specified index. Specifying 3 means that

DB2 is to collect frequent values on the concatenation of the first three key columns. The **default** is **1**, which means DB2 is to collect frequent values on the first key column of the index.

COUNT
Indicates the number of frequent values that are to be collected. Specifying 15 means that DB2 is to collect 15 frequent values from the specified key columns. The **default** is **10**.

SORTDEVT *device-type*
Specifies the device type for temporary data sets that are to be dynamically allocated by DFSORT. For *device-type*, specify any device that is valid on the DYNALLOC parameter of the SORT or OPTION options for DFSORT. See *DFSORT Application Programming: Guide* for more information.

SORTNUM *integer*
Specifies the number of temporary data sets that are to be dynamically allocated when collecting statistics for a data-partitioned secondary index. If you omit SORTDEVT, SORTNUM is ignored. If you use SORTDEVT and omit SORTNUM, no value is passed to DFSORT; DFSORT uses its own default.

integer is the number of temporary data sets that can range from 2 to 255.

You need at least two sort work data sets for each sort. The SORTNUM value applies to each sort invocation in the utility. For example, if there are three indexes, SORTKEYS is specified, there are no constraints limiting parallelism, and SORTNUM is specified as 8, then a total of 24 sort work data sets will be allocated for a job.

Each sort work data set consumes both above the line and below the link virtual storage, so if you specify too high a value for SORTNUM, the utility may decrease the degree of parallelism due to virtual storage constraints, and possibly decreasing the degree down to one, meaning no parallelism.

Important: The SORTNUM keyword will not be considered if ZPARM UTSORTAL is set to YES and IGNSORTN is set to YES.

UPDATE
Indicates whether the collected statistics are to be inserted into the catalog tables. UPDATE also allows you to select statistics that are used for access path selection or statistics that are used by database administrators.

ALL Indicates that all collected statistics are to be updated in the catalog. The **default** is **ALL**.

ACCESSPATH
Indicates that only the catalog table columns that provide statistics that are used for access path selection are to be updated.

SPACE
Indicates that only the catalog table columns that provide statistics to help the database administrator to assess the status of a particular table space or index are to be updated.

NONE
Indicates that catalog tables are not to be updated with the collected statistics. This option is valid only when REPORT YES is specified.

HISTORY
Indicates that all catalog table inserts or updates to the catalog history tables are to be recorded.

The default is supplied by the specified value in STATISTICS HISTORY on panel DSNTIPO.

ALL Indicates that all collected statistics are to be updated in the catalog history tables.

ACCESSPATH
Indicates that only the catalog history table columns that provide statistics used for access path selection are to be updated.

SPACE
Indicates that only space-related catalog statistics are to be updated in catalog history tables.

NONE
Indicates that catalog history tables are not to be updated with the collected statistics.

FORCEROLLUP
Specifies whether aggregation or rollup of statistics are to take place when RUNSTATS is executed even when some parts are empty. This option enables the optimizer to select the best access path.

YES Indicates that forced aggregation or rollup processing is to be done, even though some parts might not contain data.

NO Indicates that aggregation or rollup is to be done only if data is available for all parts.

If data is not available for all parts and if the installation value for STATISTICS ROLLUP on panel DSNTIPO is set to NO, DSNU623I message is issued.

WORKDDN(*ddname***)**
ddname specifies the DD statement for the unload data set.

ddname
Is the DD name of the temporary work file for build input. The **default** is **SYSUT1**.

The WORKDDN keyword specifies either a DD name or a TEMPLATE name from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses DD name. For more information about TEMPLATE specifications, see Chapter 31, “TEMPLATE,” on page 593.

Even though WORKDDN is an optional keyword, a DD statement for the unload output data set is required in the JCL. If you do not specify WORKDDN, or if you specify it without *ddname*, the JCL must have a DD statement with the name SYSUT1. If *ddname* is given, you must provide a DD statement or TEMPLATE that matches the DD name.

PREFORMAT
Specifies that the remaining pages are to be preformatted up to the high-allocated RBA in the index space. The preformatting occurs after the index is built.

PREFORMAT can operate on an entire index space, or on a partition of a partitioned index space.

PREFORMAT is ignored if you specify UNLOAD ONLY.

For more information about PREFORMAT, see "Improving performance with LOAD or REORG PREFORMAT" on page 254.

Instructions for running REORG INDEX

To run REORG INDEX, you must:

1. Read "Before running REORG INDEX" in this section.
2. Prepare the necessary data sets, as described in "Data sets that REORG INDEX uses" on page 403.
3. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for REORG INDEX, see "Sample REORG INDEX control statements" on page 415.)
4. Prepare a utility control statement that specifies the options for the tasks you want to perform, as described in "Instructions for specific tasks" on page 406.
5. Check the compatibility table in "Concurrency and compatibility for REORG INDEX" on page 413 if you want to run other jobs concurrently on the same target objects.
6. Plan for restart if the REORG job doesn't complete, as described in "Terminating or restarting REORG INDEX" on page 411.
7. Run REORG by one of the methods that are described inChapter 3, "Invoking DB2 online utilities," on page 15.

Before running REORG INDEX

Region size: The recommended minimum region size is 4096 KB.

Restart-pending status and SHRLEVEL CHANGE: If you specify SHRLEVEL CHANGE, REORG drains the write claim class near the end of REORG processing. In a data sharing environment, if a data sharing member fails and that member has restart-pending status for a target page set, the drain can fail. You must postpone running REORG with SHRLEVEL CHANGE until all restart-pending statuses have been removed. You can use the DISPLAY GROUP command to determine whether a member's status is FAILED. You can use the DISPLAY DATABASE command with the LOCKS option to determine if locks are held.

Data sharing considerations for REORG: You must not execute REORG on an object if another DB2 subsystem holds retained locks on the object or has long-running noncommitting applications that use the object. You can use the DISPLAY GROUP command to determine whether a member's status is ″FAILED.″ You can use the DISPLAY DATABASE command with the LOCKS option to determine if locks are held.

RECOVER-pending and REBUILD-pending status: You cannot reorganize an index if any partition of the index is in the RECOVER-pending status or in the REBUILD-pending status. Similarly, you cannot reorganize a single index partition if it is in the RECOVER-pending status or in the REBUILD-pending status.

The RECOVER-pending restrictive state is:

RECP The index space or partition is in a RECOVER-pending status. A single logical partition in RECP does not restrict access to other logical partitions that are not in RECP. You can reset RECP by recovering only the single logical partition.

The REBUILD-pending restrictive states are:

RBDP REBUILD-pending status is set on a physical or logical index partition. The individual physical or logical partition is inaccessible; you must rebuild the object using the REBUILD INDEX utility.

PSRBD
Page set REBUILD-pending (PSRBD) is set for nonpartitioning indexes. The entire index space is inaccessible; you must rebuild the object by using the REBUILD INDEX utility.

RBDP*
A REBUILD-pending status is set only on logical partitions of nonpartitioning indexes. The entire index is inaccessible, but it is made available again when you rebuild the affected partitions by using the REBUILD INDEX utility.

For information about resetting the REBUILD-pending and RECOVER-pending states, see Table 170 on page 857 and Table 171 on page 857.

CHECK-pending status: You cannot reorganize an index when the data is in the CHECK-pending status. See Chapter 8, "CHECK DATA," on page 59 for more information about resetting the CHECK-pending status.

Data sets that REORG INDEX uses

Table 63 lists the data sets that REORG uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 63. Data sets that REORG INDEX uses

Data set	Description	Required?
SYSIN	Input data set that contain the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
STPRIN01	A data set that contains messages from DFSORT (usually, SYSOUT or DUMMY). This data set is used when frequency statistics are collected on DPSI's or when TABLESPACE TABLE COLGROUP FREQVAL is specified	No [1]
Work data set	A temporary data set for unload output and build input. Specify the DD or template name with the WORKDDN option of the utility control statement. The default DD name is SYSUT1.	Yes
Sort work data sets	Temporary data sets for sort input and output when collecting inline statistics on at least one data-partitioned secondary index. The DD names have the form ST01WK*nn*.	No[1, 2, 3]

Table 63. Data sets that REORG INDEX uses (continued)

Data set	Description	Required?
Notes: 1. Required when collecting inline statistics on at least one data-partitioned secondary index. 2. If the DYNALLOC parm of the SORT program is not turned on, you need to allocate the data set. Otherwise, DFSORT dynamically allocates the temporary data set. 3. It is recommended that you use dynamic allocation by specifying SORTDEVT in the utility statement because dynamic allocation reduces the maintenance required of the utility job JCL.		

The following objects are named in the utility control statement and do not require DD statements in the JCL:

Index Object to be reorganized.

Calculating the size of the work data sets: When reorganizing an index space, you need a non-DB2 sequential work data set. That data set is identified by the DD statement that is named in the WORKDDN option. During the UNLOAD phase, the index keys and the data pointers are unloaded to the work data set. This data set is used to build the index. It is required only during the execution of REORG.

Use the following formula to calculate the approximate size (in bytes) of the WORKDDN data set SYSUT1:

```
size = number of keys x (key length + 8)
```

where

```
number of keys = #tablerows
```

Calculating the size of the sort work data sets: To calculate the approximate size (in bytes) of the ST01WK*nn* data set, use the following formula:

2 ×(*maximum record length* × *numcols* × (*count* + 2) × number of indexes)

The variables in the preceding formula have the following values:

maximum record length
Maximum record length of the SYSCOLDISTSTATS record that is processed when collecting frequency statistics (You can obtain this value from the RECLENGTH column in SYSTABLES.)

numcols
Number of key columns to concatenate when you collect frequent values from the specified index.

count Number of frequent values that DB2 is to collect.

DB2 utilities uses DFSORT to perform sorts. Sort work data sets cannot span volumes. Smaller volumes require more sort work data sets to sort the same amount of data; therefore, large volume sizes can reduce the number of needed sort work data sets. It is recommended that at least 1.2 times the amount of data to be sorted be provided in sort work data sets on disk. For more information about DFSORT, see *DFSORT Application Programming Guide*.

Shadow data sets

When you execute the REORG INDEX utility with SHRLEVEL REFERENCE or SHRLEVEL CHANGE, the utility uses shadow data sets.

For user-managed data sets, you must preallocate the shadow data sets before you execute REORG INDEX with SHRLEVEL REFERENCE or SHRLEVEL CHANGE. If an index or partitioned index resides in DB2-managed data sets and shadow data sets do not already exist when you execute REORG INDEX, DB2 creates the shadow data sets. At the end of REORG processing, the DB2-managed shadow data sets are deleted. You can create the shadows ahead of time for DB2-managed data sets.

Shadow data set names: Each shadow data set must have the following name:

catname.DSNDB*x*.*dbname*.*psname*.*y*0001.*Lnnn*

In the preceding name, the variables have the following meanings:

variable	meaning
catname	The VSAM catalog name or alias
x	C or D
dbname	Database name
psname	Table space name or index name
y	I or J
Lnnn	Partition identifier. Use one of the following values: • A001 through A999 for partitions 1 through 999 • B000 through B999 for partitions 1000 through 1999 • C000 through C999 for partitions 2000 through 2999 • D000 through D999 for partitions 3000 through 3999 • E000 through E996 for partitions 4000 through 4096

To determine the names of existing shadow data sets, execute one of the following queries against the SYSTABLEPART or SYSINDEXPART catalog tables:

```
SELECT DBNAME, TSNAME, IPREFIX
  FROM SYSIBM.SYSTABLEPART
  WHERE DBNAME = 'dbname' AND TSNAME = 'psname';
SELECT DBNAME, IXNAME, IPREFIX
  FROM SYSIBM.SYSINDEXES X, SYSIBM.SYSINDEXPART Y
  WHERE X.NAME = Y.IXNAME AND X.CREATOR = Y.IXCREATOR
  AND X.DBNAME = 'dbname' AND X.INDEXSPACE = 'psname';
```

Defining shadow data sets: Consider the following actions when you preallocate the data sets:

- Allocate the shadow data sets according to the rules for user-managed data sets.
- Define the shadow data sets as LINEAR.
- Use SHAREOPTIONS(3,3).
- Define the shadow data sets as EA-enabled if the original table space or index space is EA-enabled.
- Allocate the shadow data sets on the volumes that are defined in the storage group for the original table space or index space.

If you specify a secondary space quantity, DB2 does not use it. Instead, DB2 uses the SECQTY value for the table space or index space.

Recommendation: Use the MODEL option, which causes the new shadow data set to be created like the original data set. This method is shown in the following example:

```
DEFINE CLUSTER +
   (NAME('catname.DSNDBC.dbname.psname.x0001.L001') +
   MODEL('catname.DSNDBC.dbname.psname.y0001.L001')) +
   DATA                                              +
   (NAME('catname.DSNDBD.dbname.psname.x0001.L001') +
   MODEL('catname.DSNDBD.dbname.psname.y0001.L001') )
```

DB2 treats preallocated shadow data sets as DB2-managed data sets.

Creating shadow data sets for indexes: When you preallocate data sets for indexes, create the shadow data sets as follows:

- Create shadow data sets for the partition of the table space and the corresponding partition in each partitioning index and data-partitioned secondary index.
- Create a shadow data set for logical partitions of nonpartitioned secondary indexes.

Use the same naming scheme for these index data sets as you use for other data sets that are associated with the base index, except use J0001 instead of I0001. For more information about this naming scheme, see the information about the shadow data set naming convention at the beginning of this section.

Estimating the size of shadow data sets: If you do not change the value of FREEPAGE or PCTFREE, the amount of space that is required for a shadow data set is approximately comparable to the amount of space that is required for the original data set. For more information about calculating the size of data sets, see "Data sets that REORG INDEX uses" on page 403.

Creating the control statement

Create the utility control statement for the REORG INDEX job. See "Syntax and options of the REORG INDEX control statement" on page 390 for REORG syntax and option descriptions. See "Sample REORG INDEX control statements" on page 415 for examples of REORG usage.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in your utility control statement:

"Determining when an index requires reorganization" on page 407
"Using the LEAFDISTLIMIT and REPORTONLY options to determine when reorganization is needed" on page 407
"Specifying access with SHRLEVEL" on page 408
"Considerations for fallback recovery" on page 409
"Changing data set definitions" on page 409
"Temporarily interrupting REORG" on page 409
"Improving performance" on page 410
"Improving performance with LOAD or REORG PREFORMAT" on page 254
"The effect of LOAD on index version numbers" on page 273

Determining when an index requires reorganization

Product-sensitive Programming Interface

Use the following query to identify user-created indexes and DB2 catalog indexes that you should consider reorganizing with the REORG INDEX utility:

```
EXEC SQL
SELECT IXNAME, IXCREATOR
 FROM SYSIBM.SYSINDEXPART
   WHERE LEAFDIST > 200
ENDEXEC
```

End of Product-sensitive Programming Interface

Be aware that using a LEAFDIST value of more than 200 as an indicator of a disorganized index is merely a rough guideline for general cases. This guidance is not absolute. In some cases, 200 is an acceptable value for LEAFDIST. For example, with FREEPAGE 0 and index page splitting, the LEAFDIST value can climb sharply. In this case, a LEAFDIST value that exceeds 200 can be acceptable.

Product-sensitive Programming Interface

After you run RUNSTATS, issuing the following SQL statement provides the average distance (multiplied by 100) between successive leaf pages during sequential access of the ZZZ index.

```
EXEC SQL
SELECT LEAFDIST
  FROM SYSIBM.SYSINDEXPART
  WHERE IXCREATOR = 'index_creator_name'
  AND IXNAME = 'index_name'
ENDEXEC
```

End of Product-sensitive Programming Interface

An increase in the LEAFDIST value over time probably indicates that the index needs to be reorganized. The optimal value of the LEAFDIST catalog column is zero. However, immediately after you run the REORG and RUNSTATS utilities, LEAFDIST might be greater than zero as a result of empty pages for FREEPAGE and non-leaf pages.

For specific REORG threshold numbers, see Part 5 of *DB2 Administration Guide*.

Using the LEAFDISTLIMIT and REPORTONLY options to determine when reorganization is needed

Product-sensitive Programming Interface

You can determine when to run REORG for indexes by using the LEAFDISTLIMIT option. If you specify the REPORTONLY option, REORG produces a report that indicates whether a REORG is recommended; a REORG is not performed.

When you specify the LEAFDISTLIMIT option with the REPORTONLY option, REORG produces a report with one of the following return codes:

1 No limit met; no REORG performed or recommended.
2 REORG performed or recommended.

Alternatively, information from the SYSINDEXPART catalog table can tell you which indexes qualify for reorganization.

End of Product-sensitive Programming Interface

Specifying access with SHRLEVEL

For reorganizing an index or a partition of a index, the SHRLEVEL option lets you choose the level of access that you have to your data during reorganization:

- REORG with SHRLEVEL NONE, the default, reloads the reorganized data into the original area that is being reorganized. Applications have read-only access during unloading and no access during reloading. SHRLEVEL NONE is the only access level that resets REORG-pending status.
- REORG with SHRLEVEL REFERENCE reloads the reorganized data into a new (shadow) copy of the area that is being reorganized. Near the end of reorganization, DB2 switches applications' future access from the original to the shadow copy. For SHRLEVEL REFERENCE, applications have read-only access during unloading and reloading, and a brief period of no access during switching.
- REORG with SHRLEVEL CHANGE reloads the reorganized data into a shadow copy of the area that is being reorganized. Applications can read from and write to the original area, and DB2 records the writing in the log. DB2 then reads the log and applies it to the shadow copy to bring the shadow copy up to date. This step executes iteratively, with each iteration processing a sequence of log records. Near the end of reorganization, DB2 switches applications' future access from the original to the shadow copy. Applications have read-write access during unloading and reloading, a brief period of read-only access during the last iteration of log processing, and a brief period of no access during switching.

Log processing with SHRLEVEL CHANGE: When you specify SHRLEVEL CHANGE, DB2 processes the log to update the shadow copy. This step executes iteratively. The first iteration processes the log records that accumulated during the previous iteration. The iterations continue until one of these conditions is met:

- DB2 estimates that the time to perform the log processing in the next iteration will be less than or equal to the time that is specified by MAXRO. If this condition is met, the next iteration is the last.
- DB2 estimates that the switch phase will not start by the deadline specified by DEADLINE. If this condition is met, DB2 terminates reorganization.
- The number of log records that the next iteration will process is not sufficiently lower than the number of log records that were processed in the previous iteration. If this condition is met but the first two conditions are not, DB2 sends message DSNU377I to the console. DB2 continues log processing for the length of time that is specified by DELAY and then performs the action specified by LONGLOG.

Operator actions: LONGLOG specifies the action that DB2 is to perform if log processing is not occurring quickly enough. See "Option descriptions" on page 393 for a description of the LONGLOG options. If the operator does not respond to the console message DSNU377I, the LONGLOG option automatically goes into effect. You can take one of the following actions:

- Execute the START DATABASE(db) SPACENAM(ts)... ACCESS(RO) command and the QUIESCE utility to drain the write claim class. DB2 performs the last iteration, if MAXRO is not DEFER. After the QUIESCE, you should also execute the ALTER UTILITY command, even if you do not change any REORG parameters.

- Execute the START DATABASE(db) SPACENAM(ts)... ACCESS(RO) command and the QUIESCE utility to drain the write claim class. Then, after reorganization has made some progress, execute the START DATABASE(db) SPACENAM(ts)... ACCESS(RW) command. This action increases the likelihood that log processing can improve. After the QUIESCE, you should also execute the ALTER UTILITY command, even if you do not change any REORG parameters.
- Execute the ALTER UTILITY command to change the value of MAXRO. Changing it to a huge positive value, such as 9999999, causes the next iteration to be the last iteration.
- Execute the ALTER UTILITY command to change the value of LONGLOG.
- Execute the TERM UTILITY command to terminate reorganization.
- Adjust the amount of buffer space that is allocated to reorganization and to applications. This adjustment can increase the likelihood that log processing improve after adjusting the space, you should also execute the ALTER UTILITY command, even if you do not change any REORG parameters.
- Adjust the scheduling priorities of reorganization and applications. This adjustment can increase the likelihood that log processing improve. After adjusting the priorities, you should also execute the ALTER UTILITY command, even if you do not change any REORG parameters.

DB2 does not take the action specified in the LONGLOG phrase if any one of these events occurs before the delay expires:

- An ALTER UTILITY command is issued.
- A TERM UTILITY command is issued.
- DB2 estimates that the time to perform the next iteration is likely to be less than or equal to the time specified on the MAXRO keyword.
- REORG terminates for any reason (including the deadline).

Considerations for fallback recovery

Successful REORG INDEX processing inserts a SYSCOPY row with ICTYPE='W' for an index that was defined with COPY YES. REORG also places a reorganized index in informational COPY-pending status. You should take a full image copy of the index after the REORG job completes to create a valid point of recovery.

Changing data set definitions

If the index space is defined by storage groups, space allocation is handled by DB2 and data set definitions cannot be altered during the reorganization process. DB2 deletes and redefines the necessary data sets to reorganize the object.

For REORG with SHRLEVEL REFERENCE or CHANGE, you can use the ALTER STOGROUP command to change the characteristics of a DB2-managed data set. You can effectively change the characteristics of a user-managed data set by specifying the desired new characteristics when creating the shadow data set; see "Shadow data sets" on page 405 for more information about shadow data sets. In particular, placing the original and shadow data sets on different disk volumes might reduce contention and thus improve the performance of REORG and the performance of applications during REORG execution.

Temporarily interrupting REORG

You can temporarily pause REORG. If you specify UNLOAD PAUSE, REORG pauses after unloading the index space into the work data set. The job completes with return code 4. You can restart REORG by using the phase restart or current restart. The REORG statement must not be altered.

The SYSIBM.SYSUTIL record for the REORG INDEX utility remains in ″stopped″ status until REORG is restarted or terminated.

While REORG is interrupted by PAUSE, you can re-define the table space attributes for user defined table spaces. PAUSE is not required for STOGROUP-defined table spaces. Attribute changes are done automatically by a REORG following an ALTER INDEX.

Improving performance

To improve REORG performance, run REORG concurrently on separate partitions of a partitioned index space. The processor time for running REORG INDEX on partitions of a partitioned index is approximately the same as the time for running a single REORG index job. The elapsed time is a fraction of the time for running a single REORG job on the entire index

When to use SHRLEVEL CHANGE: Schedule REORG with SHRLEVEL CHANGE when the rate of writing is low and transactions are short. Avoid scheduling REORG with SHRLEVEL CHANGE when low-tolerance applications are executing.

When to use DRAIN_WAIT: The DRAIN_WAIT option provides improved control over the time online REORG waits for drains. Also, because the DRAIN_WAIT is the aggregate time that online REORG is to wait to perform a drain on a table space and associated indexes, the length of drains is more predictable than it is when each partition and index has its own individual waiting-time limit.

By specifying a short delay time (less than the system timeout value, IRLMRWT), you can reduce the impact on applications by reducing time-outs. You can use the RETRY option to give the online REORG INDEX utility chances to complete successfully. If you do not want to use RETRY processing, you can still use DRAIN_WAIT to set a specific and more consistent limit on the length of drains.

RETRY allows an online REORG that is unable to drain the objects it requires to try again after a set period (RETRY_DELAY). If the drain fails in the SWITCH phase, the objects remain in their original state (read-only mode for SHRLEVEL REFERENCE or read-write mode for SHRLEVEL CHANGE). Likewise, objects will remain in their original state if the drain fails in the LOG phase.

Because application SQL statements can queue behind any unsuccessful drain that the online REORG has tried, define a reasonable delay before you retry to allow this work to complete; the default is 5 minutes.

When the default DRAIN WRITERS is used with SHRLEVEL CHANGE and RETRY, multiple read-only log iterations can occur. Because online REORG can have to do more work when RETRY is specified, multiple or extended periods of restricted access might occur. Applications that run with REORG must perform frequent commits. During the interval between retries, the utility is still active; consequently, other utility activity against the table space and indexes is restricted.

When you perform a table space REORG and specify both RETRY and SHRLEVEL CHANGE, the size of the copy that REORG takes might increase.

Recommendation: Run online REORG during light periods of activity on the table space or index.

Terminating or restarting REORG INDEX

This section contains information about how to terminate and restart REORG INDEX.

Terminating REORG INDEX

If you terminate REORG with the TERM UTILITY command during the UNLOAD phase, objects have not yet been changed, and you can rerun the job.

If you terminate REORG with the TERM UTILITY command during the build phase, the behavior depends on the SHRLEVEL option:

- For SHRLEVEL NONE, the index is left in RECOVER-pending status. After you recover the index, rerun the REORG job.
- For SHRLEVEL REFERENCE or CHANGE, the index keys are reloaded into a shadow index, so the original index has not been affected by REORG. You can rerun the job.

If you terminate REORG with the TERM UTILITY command during the log phase, the index keys are reloaded into a shadow index, so the original index has not been affected by REORG. You can rerun the job.

If you terminate REORG with the TERM UTILITY command during the switch phase, all data sets that were renamed to their shadow counterparts are renamed back, so the objects are left in their original state. You can rerun the job. If a problem occurs in renaming to the original data sets, the objects are left in RECOVER-pending status. You must recover the index.

The REORG-pending status is not reset until the UTILTERM execution phase. If the REORG INDEX utility abnormally terminates or is terminated, the objects are left in RECOVER-pending status. See Appendix C, “Advisory or restrictive states,” on page 853 for information about resetting either status.

Table 64 lists any restrictive states that are set based on the phase in which REORG INDEX terminated.

Table 64. Restrictive states set based on the phase in which REORG INDEX terminated

Phase	Effect on restrictive status
UNLOAD	No effect.
BUILD	Sets REBUILD-pending (RBDP) status at the beginning of the build phase, and resets RBDP at the end of the phase. SHRLEVEL NONE places an index that was defined with the COPY YES attribute in RECOVER pending (RECP) status.
LOG	No effect.
SWITCH	Under certain conditions, if TERM UTILITY is issued, it must complete successfully; otherwise, objects might be placed in RECP status or RBDP status. For SHRLEVEL REFERENCE or CHANGE, sets the RECP status if the index was defined with the COPY YES attribute at the beginning of the switch phase, and resets RECP at the end of the phase. If the index was defined with COPY NO, this phase sets the index in RBDP status at the beginning of the phase, and resets RBDP at the end of the phase.

Restarting REORG INDEX

Table 65 on page 412 provides information about restarting REORG INDEX.

If you restart REORG in the outlined phase, it re-executes from the beginning of the phase. DB2 always uses RESTART(PHASE) by default unless you restart the job in the UNLOAD phase. In this case, DB2 uses RESTART(CURRENT) by default.

If REORG abnormally terminates or a system failure occurs while it is in the UTILTERM phase, you must restart the job with RESTART(PHASE).

For each phase of REORG and for each type of REORG INDEX (with SHRLEVEL NONE, with SHRLEVEL REFERENCE, and with SHRLEVEL CHANGE), the table indicates the types of restart that are allowed (CURRENT and PHASE). None indicates that no restart is allowed. The ″Data sets required″ column lists the data sets that must exist to perform the specified type of restart in the specified phase.

Table 65. REORG INDEX utility restart information

Phase	Type of restart allowed for SHRLEVEL NONE	Type of restart allowed for SHRLEVEL REFERENCE	Type of restart allowed for SHRLEVEL CHANGE	Data sets required	Notes
UNLOAD	CURRENT, PHASE	CURRENT, PHASE	None	SYSUT1	
BUILD	CURRENT, PHASE	CURRENT, PHASE	None	SYSUT1	1
LOG	Phase does not occur	Phase does not occur	None	None	
SWITCH	Phase does not occur	CURRENT, PHASE	CURRENT, PHASE	originals and shadows	1

Notes:

1. You can restart the utility with either RESTART or RESTART(PHASE). However, because this phase does not take checkpoints, RESTART always re-executes from the beginning of the phase.

If you restart a REORG STATISTICS job that was stopped in the BUILD phase by using RESTART CURRENT, inline statistics collection does not occur. To update catalog statistics, run the RUNSTATS utility after the restarted job completes. Restarting a REORG STATISTICS job with RESTART(PHASE) is conditional after executing UNLOAD PAUSE. To determine if catalog table statistics are to be updated when you restart a REORG STATISTICS job, see Table 66. This table lists whether or not statistics are updated based on the execution phase and whether the job is restarted with RESTART(CURRENT) or RESTART(PHASE).

Table 66. Whether statistics are updated when REORG INDEX STATISTICS jobs are restarted in certain phases

Phase	RESTART CURRENT	RESTART PHASE
UTILINIT	No	Yes
UNLOAD	No	Yes
BUILD	No	Yes

For instructions on restarting a utility job, see Chapter 3, “Invoking DB2 online utilities,” on page 15.

Restarting REORG after an out-of-space condition: See “Restarting after the output data set is full” on page 45 for guidance in restarting REORG from the last commit point after receiving an out-of-space condition.

Concurrency and compatibility for REORG INDEX

DB2 treats individual index partitions as distinct target objects. Utilities that operate on different partitions of the same index space are compatible.

Table 67 shows which claim classes REORG INDEX drains and any restrictive state the utility sets on the target object. The target is an index or index partition.

Table 67. Claim classes of REORG INDEX operations

Phase	REORG INDEX SHRLEVEL NONE	REORG INDEX SHRLEVEL REFERENCE	REORG INDEX SHRLEVEL CHANGE
UNLOAD	DW/UTRO	DW/UTRO	CR/UTRW
BUILD	DA/UTUT	none	none
Last iteration of LOG	n/a	DA/UTUT[1]	DW/UTRO
SWITCH	n/a	DA/UTUT	DA/UTUT

Legend:
- CR: Claim the read claim class.
- DA: Drain all claim classes, no concurrent SQL access.
- DR: Drain the repeatable read class, no concurrent access for SQL repeatable readers.
- DW: Drain the write claim class, concurrent access for SQL readers.
- UTRO: Utility restrictive state, read only access allowed.
- UTUT: Utility restrictive state, exclusive control.
- none: Any claim, drain, or restrictive state for this object does not change in this phase.

Notes:

1. Applicable if you specified DRAIN ALL.

Table 68 shows which utilities can run concurrently with REORG INDEX on the same target object. The target object can be an index space or a partition. If compatibility depends on particular options of a utility, that is also shown. REORG INDEX does not set a utility restrictive state if the target object is an index on DSNDB01.SYSUTILX.

Table 68. Compatibility of REORG INDEX with other utilities

Action	REORG INDEX SHRLEVEL NONE, REFERENCE, or CHANGE
CHECK DATA	No
CHECK INDEX	No
CHECK LOB	Yes
COPY INDEXSPACE	No
COPY TABLESPACE	Yes
DIAGNOSE	Yes
LOAD	No
MERGECOPY	Yes
MODIFY	Yes
QUIESCE	No
REBUILD INDEX	No
RECOVER INDEX	No

Table 68. Compatibility of REORG INDEX with other utilities (continued)

Action	REORG INDEX SHRLEVEL NONE, REFERENCE, or CHANGE
RECOVER INDEXSPACE	No
RECOVER TABLESPACE (with no options)	Yes
RECOVER TABLESPACE ERROR RANGE	Yes
RECOVER TABLESPACE TOCOPY or TORBA	No
REORG INDEX SHRLEVEL NONE, REFERENCE, or CHANGE	No
REORG TABLESPACE SHRLEVEL NONE UNLOAD CONTINUE or PAUSE, REORG SHRLEVEL REFERENCE, or REORG SHRLEVEL CHANGE	No
REORG TABLESPACE SHRLEVEL NONE UNLOAD ONLY or EXTERNAL with cluster index	No
REORG TABLESPACE SHRLEVEL NONE UNLOAD ONLY or EXTERNAL without cluster index	Yes
REPAIR LOCATE INDEX PAGE REPLACE	No
REPAIR LOCATE KEY	No
REPAIR LOCATE RID DELETE	No
REPAIR LOCATE RID DUMP, VERIFY, or REPLACE	Yes
REPAIR LOCATE TABLESPACE PAGE REPLACE	Yes
REPORT	Yes
RUNSTATS INDEX	No
RUNSTATS TABLESPACE	Yes
STOSPACE	Yes
UNLOAD	Yes

To run on SYSIBM.DSNLUX01 or SYSIBM.DSNLUX02, REORG INDEX must be the only utility in the job step and the only utility that is running in the DB2 subsystem.

Reviewing REORG INDEX output

The output from REORG INDEX consists of a reorganized index or index partition. Table 69 summarizes the results of REORG INDEX based upon what you specified.

Table 69. Summary of the results of REORG INDEX

Specification	Results
REORG INDEX	Entire index (all partitions of a partitioned index)
REORG INDEX PART *n*	Part *n* of partitioned index

When reorganizing an index, REORG leaves free pages and free space on each page in accordance with the current values of the FREEPAGE and PCTFREE parameters. (You can set those values by using the CREATE INDEX or ALTER INDEX statement.) REORG leaves one free page after reaching the FREEPAGE limit for each table in the index space.

Catalog updates: REORG INDEX updates SYSINDEXPART OLDEST_VERSION and SYSINDEXES OLDEST_VERSION (if applicable).

The effect of REORG INDEX on index version numbers

DB2 stores the range of used index version numbers in the OLDEST_VERSION and CURRENT_VERSION columns of the following catalog tables:

- SYSIBM.SYSINDEXES
- SYSIBM.SYSINDEXPART

The OLDEST_VERSION column contains the oldest used version number, and the CURRENT_VERSION column contains the current version number.

When you run REORG INDEX, the utility updates this range of used version numbers for indexes that are defined with the COPY NO attribute. REORG INDEX sets the OLDEST_VERSION column to the current version number, which indicates that only one version is active; DB2 can then reuse all of the other version numbers.

Recycling of version numbers is required when all of the version numbers are being used. All version numbers are being used when one of the following situations is true:

- The value in the CURRENT_VERSION column is one less than the value in the OLDEST_VERSION column.
- The value in the CURRENT_VERSION column is 15 and the value in the OLDEST_VERSION column is 0 or 1.

You can also run LOAD REPLACE, REBUILD INDEX, or REORG TABLESPACE to recycle version numbers for indexes that are defined with the COPY NO attribute. To recycle version numbers for indexes that are defined with the COPY YES attribute or for table spaces, run MODIFY RECOVERY.

For more information about versions and how they are used by DB2, see Part 2 of *DB2 Administration Guide*.

Sample REORG INDEX control statements

Example 1: Reorganizing an index. The following control statement specifies that the REORG INDEX utility is to reorganize index XMSGTXT1. The UNLOAD PAUSE option indicates that after the data has been unloaded, the utility is to stop. Processing can be restarted in the RELOAD phase. This option is useful if you want to redefine data sets during reorganization.

```
REORG INDEX DSN8810.XMSGTXT1
 UNLOAD PAUSE
```

Example 2: Collecting inline statistics while reorganizing an index. The following control statement specifies that REORG INDEX is to collect statistics for index XEMPL1 while reorganizing that index. The SHRLEVEL REFERENCE option indicates that during this processing, only read access is allowed on the areas that are being reorganized

```
REORG INDEX DSN8810.XEMPL1
   SHRLEVEL REFERENCE STATISTICS
```

Example 3: Updating access path statistics in the catalog and catalog history tables while reorganizing an index. The following control statement specifies that while reorganizing index IU0E0801, REORG INDEX is to also collect statistics,

collect all of the distinct values in the key column combinations, and update access path statistics in the catalog and catalog history tables. The utility is also to send any output, including space and access path statistics, to SYSPRINT.

```
REORG INDEX IUOE0801
        STATISTICS
        KEYCARD
        REPORT YES
        UPDATE ACCESSPATH
        HISTORY ACCESSPATH
```

Example 4: Reorganizing a list of indexes. In the example in Figure 76, the OPTIONS utility control statement specifies that the subsequent TEMPLATE and LISTDEF utility control statements are to run in PREVIEW mode. If the syntax of these statements is correct, DB2 expands the REORG_INDX list and the data set names in the SREC, SUT1, and SOUT templates and prints these results to the SYSPRINT data set. The second OPTIONS control statement turns off the PREVIEW mode, and the subsequent REORG INDEX job runs normally.

The REORG INDEX statement specifies that the utility is to reorganize the indexes that are included in the REORG_INDX list. The SHRLEVEL CHANGE option indicates that during this processing, read and write access is allowed on the areas that are being reorganized, with the exception of a 100-second period during the last iteration of log processing. During this time, which is specified by the MAXRO option, applications have read-only access. The WORKDDN option indicates that REORG INDEX is to use the data set that is defined by the SUT1 template. If the SWITCH phase does not begin by the deadline that is specified on the DEADLINE option, processing terminates.

```
//STEP2    EXEC DSNUPROC,UID='HUHRU257.REORGI',TIME=1440,
//         UTPROC='',
//         SYSTEM='SSTR',DB2LEV=DB2A
//SYSIN    DD *
   OPTIONS PREVIEW
   TEMPLATE SREC
            UNIT(SYSDA) DISP(NEW,CATLG,CATLG)
            DSN(HUHRU257.REORG.&ST..SREC)
   TEMPLATE SUT1
            UNIT(SYSDA) DISP(NEW,DELETE,CATLG)
            DSN(HUHRU257.REORG.&ST..SUT1)
   TEMPLATE SOUT
            UNIT(SYSDA) DISP(NEW,DELETE,CATLG)
            DSN(HUHRU257.REORG.&ST..SOUT)
   LISTDEF REORG_INDX INCLUDE INDEX ADMF001.IPHR5701
                      INCLUDE INDEX ADMF001.IXHR570*
   OPTIONS OFF
    REORG INDEX LIST REORG_INDX
      PREFORMAT
      SHRLEVEL CHANGE
      DEADLINE 2010-2-4-23.10.12
      MAXRO 100
        WORKDDN (SUT1)
/*
```

Figure 76. Example statements for job that reorganizes a list of indexes

Chapter 25. REORG TABLESPACE

The online REORG TABLESPACE utility reorganizes a table space to improve access performance and to reclaim fragmented space. In addition, the utility can reorganize a single partition or range of partitions of a partitioned table space. You can specify the degree of access to your data during reorganization, and you can collect inline statistics by using the STATISTICS keyword. If you specify REORG TABLESPACE UNLOAD EXTERNAL, the data is unloaded in a format that is acceptable to the LOAD utility of any DB2 subsystem. You can also delete rows during the REORG job by specifying the DISCARD option.

You can determine when to run REORG for non-LOB table spaces by using the OFFPOSLIMIT or INDREFLIMIT catalog query options. If you specify the REPORTONLY option, REORG produces a report that indicates whether a REORG is recommended without actually performing the REORG.These options are not applicable and are disregarded if the target object is a directory table space.

Run the REORG TABLESPACE utility on a LOB table space to help increase the effectiveness of prefetch. For a LOB table space, REORG TABLESPACE performs these actions:

- Removes imbedded free space
- Attempts to make LOB pages contiguous

A REORG of a LOB table space does not reclaim physical space.

Do not execute REORG on an object if another DB2 holds retained locks on the object or has long-running noncommitting applications that use the object. You can use the DISPLAY GROUP command to determine whether a member's status is failed. You can use the DISPLAY DATABASE command with the LOCKS option to determine if locks are held.

For a diagram of REORG TABLESPACE syntax and a description of available options, see "Syntax and options of the REORG TABLESPACE control statement" on page 420. For detailed guidance on running this utility, see "Instructions for running REORG TABLESPACE" on page 449.

Output: If the table space or partition has the COMPRESS YES attribute, the data is compressed when it is reloaded. If you specify the KEEPDICTIONARY option of REORG, the current dictionary is used; otherwise a new dictionary is built.

You can execute the REORG TABLESPACE utility on the table spaces in the DB2 catalog database (DSNDB06) and on some table spaces in the directory database (DSNDB01). It cannot be executed on any table space in the DSNDB07 database.

Table 70. summaries the results of REORG TABLESPACE according to the type of REORG specified.

Table 70. Summary of REORG TABLESPACE output

Type of REORG specified	Results
REORG TABLESPACE	Reorganizes all data and all indexes.
REORG TABLESPACE PART *n*	Reorganizes data for PART *n* of the table space and PART *n* of all partitioned indexes.

Table 70. Summary of REORG TABLESPACE output (continued)

Type of REORG specified	Results
REORG TABLESPACE PART *n:m*	Reorganizes data for PART *n* through PART *m* of the table space and PART *n* through PART *m* of all partitioned indexes.

Note: When SCOPE PENDING is also specified, the REORG TABLESPACE utility reorganizes the specified table space only if it is in REORG-pending or advisory REORG-pending status. For a partitioned table space, REORG TABLESPACE SCOPE PENDING reorganizes only the partitions that are in REORG-pending or advisory REORG-pending status.

Authorization required: To execute this utility on a user table space, you must use a privilege set that includes one of the following authorities:
- REORG privilege for the database
- DBADM or DBCTRL authority for the database
- SYSCTRL authority
- SYSADM authority

To execute this utility on a table space in the catalog or directory, you must use a privilege set that includes one of the following authorities:
- REORG privilege for the DSNDB06 (catalog) database
- DBADM or DBCTRL authority for the DSNDB06 (catalog) database
- Installation SYSOPR authority
- SYSCTRL authority
- SYSADM or Installation SYSADM authority

If you specify REORG TABLESPACE SHRLEVEL CHANGE , you must create a mapping table. You must use a privilege set that includes DELETE, INSERT, and UPDATE privileges on the mapping table. See "Before running REORG TABLESPACE" on page 449 for more information about mapping tables.

To run REORG TABLESPACE STATISTICS TABLE REPORT YES, you must use a privilege set that includes STATS privilege for the database and the SELECT privilege on the catalog tables and tables for which statistics are to be gathered. REORG TABLESPACE STATISTICS TABLE REPORT ALL does not report values from tables that the user is not authorized to see.

An authority other than installation SYSADM or installation SYSOPR can receive message `DSNT500I resource unavailable`, while trying to reorganize a table space in the catalog or directory. This message can be issued when the DSNDB06.SYSDBAUT or DSNDB06.SYSUSER catalog table space or one of the indexes is unavailable. If this problem occurs, run the REORG TABLESPACE utility again using an authorization ID with the installation SYSADM or installation SYSOPR authority.

If you use RACF access control with multilevel security and REORG TABLESPACE is to process a table space that contains a table that has multilevel security with row-level granularity, you must be identified to RACF and have an accessible valid security label. You must also meet the following authorization requirements:
- For REORG statements that include the UNLOAD EXTERNAL option, each row is unloaded only if your security label dominates the data security label. If your security label does not dominate the data security label, the row is not unloaded, but DB2 does not issue an error message.

- For REORG statements that include the DISCARD option, qualifying rows are discarded only if one of the following situations is true:
  - Write-down rules are in effect, you have write-down privilege, and your security label dominates the data's security label.
  - Write-down rules are not in effect and your security label dominates the data's security label.
  - Your security label is equivalent to the data security label.

For more information about multilevel security and security labels, see Part 3 of *DB2 Administration Guide*.

Execution phases of REORG TABLESPACE: The REORG TABLESPACE utility operates in these phases:

Phase	Description
UTILINIT	Performs initialization and setup.
UNLOAD	Unloads the table space and sorts data if a clustering index exists and the utility job includes either the SORTDATA or SHRLEVEL CHANGE options. If you specify NOSYSREC, the utility passes rows in memory to the RELOAD phase; otherwise, it writes them to a sequential data set.
RELOAD	Reloads data from the sequential data set into the table space and creates full image copies if you specify COPYDDN, RECOVERYDDN, SHRLEVEL REFERENCE, or SHRLEVEL CHANGE. A subtask sorts the index keys. The utility also updates table and table space statistics.
SORT	Sorts index keys. The sorted keys are passed in memory to the BUILD phase.
BUILD	Builds indexes and updates index statistics.
SORTBLD	If parallel index build occurs, all activities that normally occur in both the SORT and BUILD phases occur in the SORTBLD phase instead. For more information about when parallel index build occurs, see"Building indexes in parallel for REORG TABLESPACE" on page 472.
LOG	Processes the log iteratively and appends changed pages to the full image copies. This phase occurs only if you specify SHRLEVEL CHANGE.
SWITCH	Switches access to shadow copy of table space or partition. This phase occurs only if you specify SHRLEVEL REFERENCE or CHANGE.
BUILD2	Corrects nonpartitioning indexes if you specify REORG TABLESPACE PART SHRLEVEL REFERENCE or CHANGE.
UTILTERM	Performs cleanup.

Execution phases of REORG TABLESPACE on a LOB table space: The REORG TABLESPACE utility operates in these phases when you run it on a LOB table space:

Phase	Description
UTILINIT	Performs initialization and setup.
REORGLOB	Rebuilds the LOB table space in place. The utility does not unload

or reload LOBs. The LOB table space is set to RECOVER-pending status at the start of processing; this status is reset when the REORGLOB phase completes. If the REORGLOB phase fails, the LOB table space remains in RECOVER-pending status.

UTILTERM Performs cleanup.

You cannot restart REORG TABLESPACE on a LOB table space in the REORGLOB phase. Before executing REORG TABLESPACE on a LOB table space that is defined with LOG NO, you should take a full image copy to ensure recoverability.

If the LOB table space is defined with LOG NO, it is left in COPY-pending status after REORG TABLESPACE completes processing.

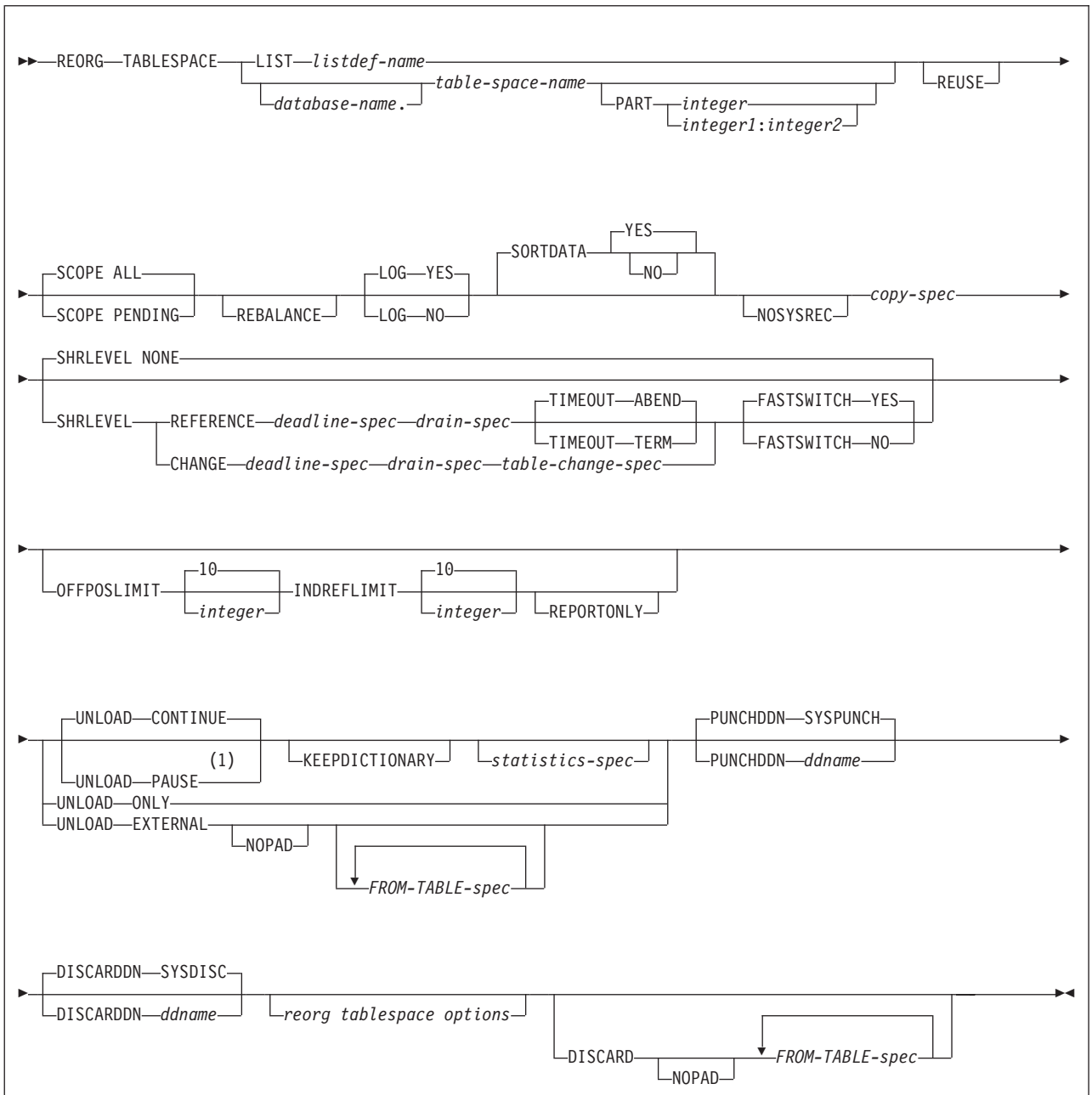
The following topics provide additional information:

- "Syntax and options of the REORG TABLESPACE control statement"
- "Instructions for running REORG TABLESPACE" on page 449
- "Concurrency and compatibility for REORG TABLESPACE" on page 480
- "Reviewing REORG TABLESPACE output" on page 484
- "After running REORG TABLESPACE" on page 484
- "Effects of running REORG TABLESPACE" on page 485
- "Sample REORG TABLESPACE control statements" on page 486

Syntax and options of the REORG TABLESPACE control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

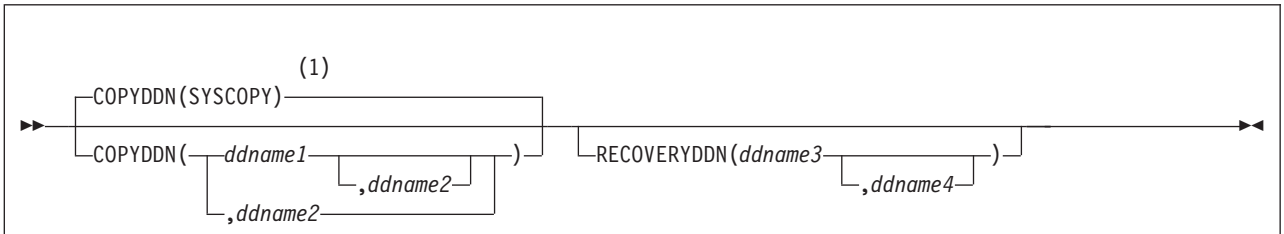
Syntax diagram



Notes:

1 You cannot use UNLOAD PAUSE with the LIST option.

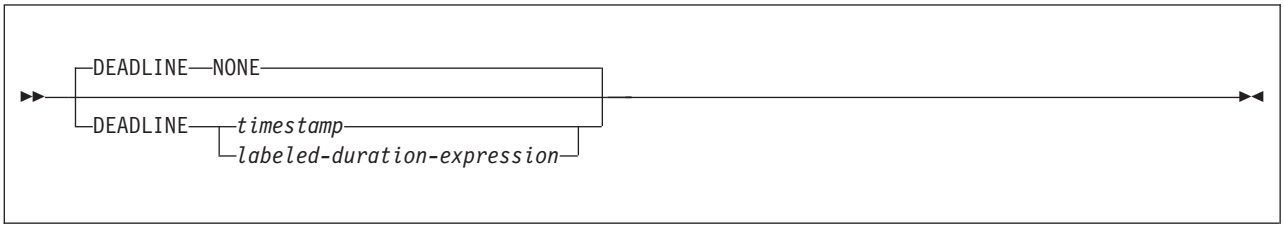
copy-spec:



Notes:

1 COPYDDN(SYSCOPY) is not the default if you specify SHRLEVEL NONE and no partitions are in REORG-pending status.

deadline-spec:



drain-spec:

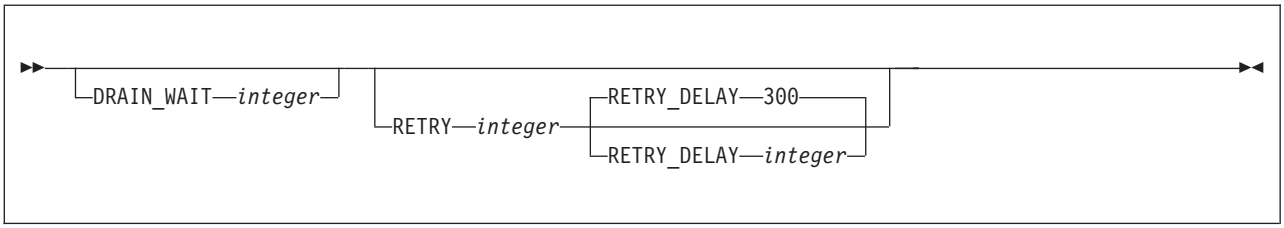
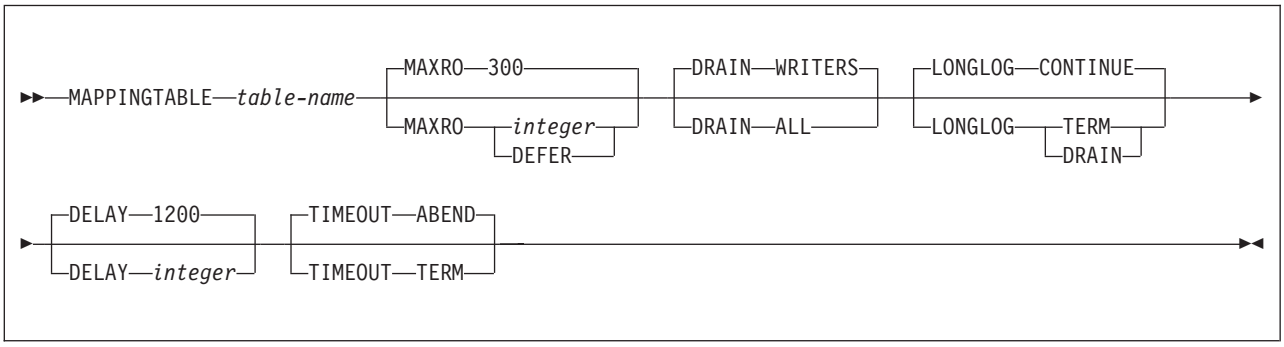
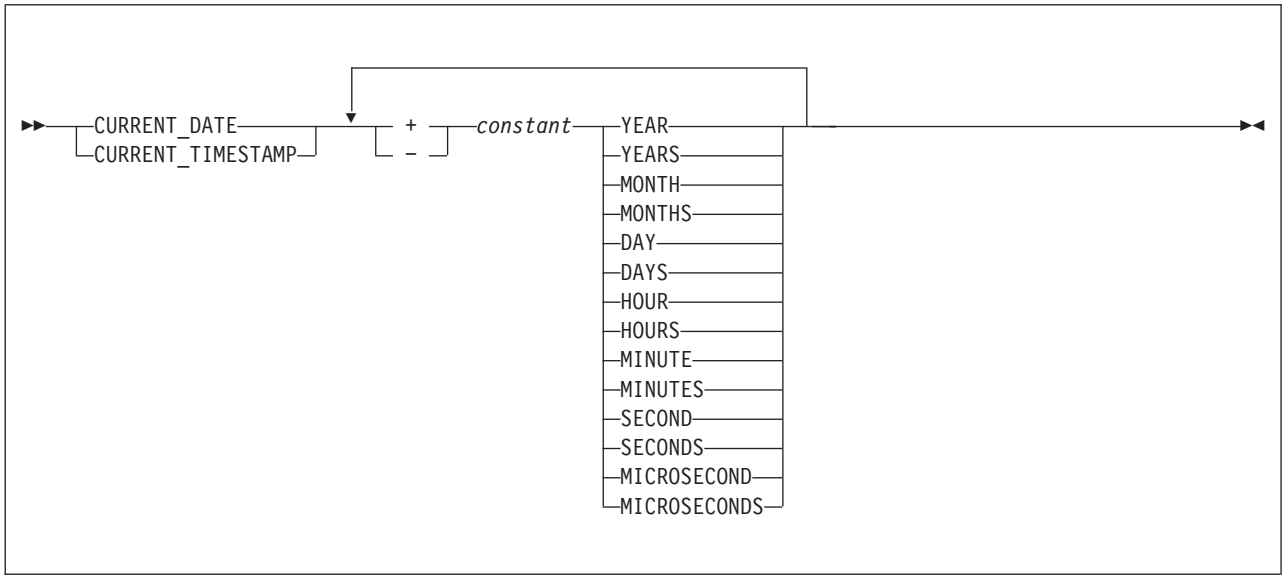


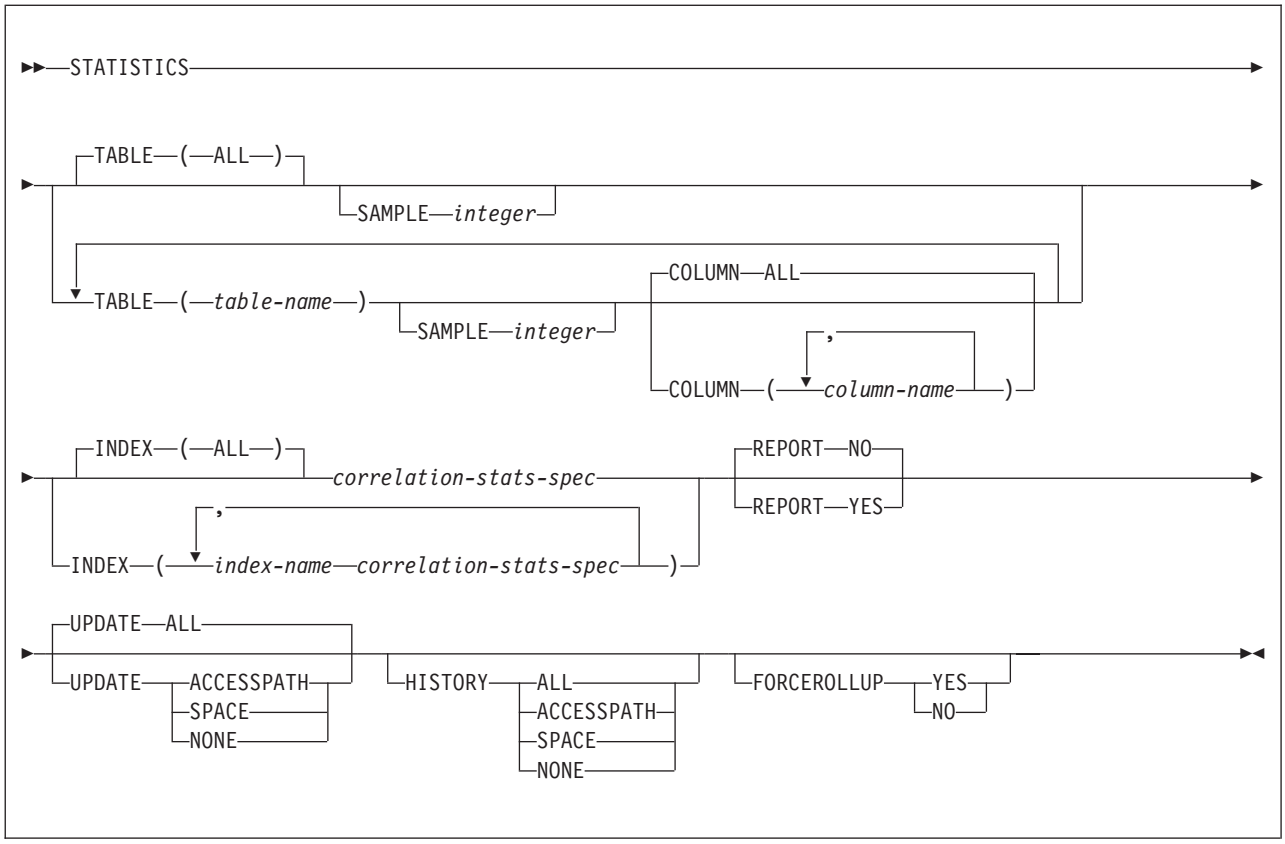
table-change-spec:



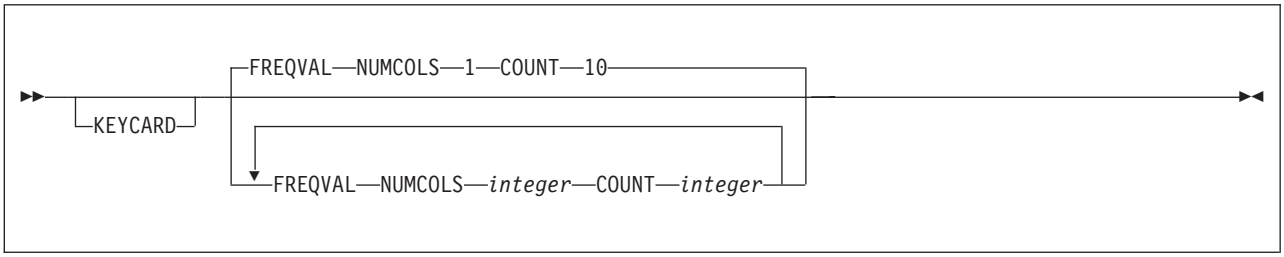
labeled-duration-expression:



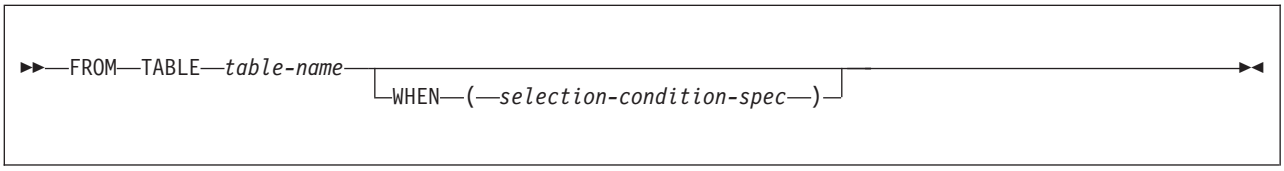
statistics-spec:



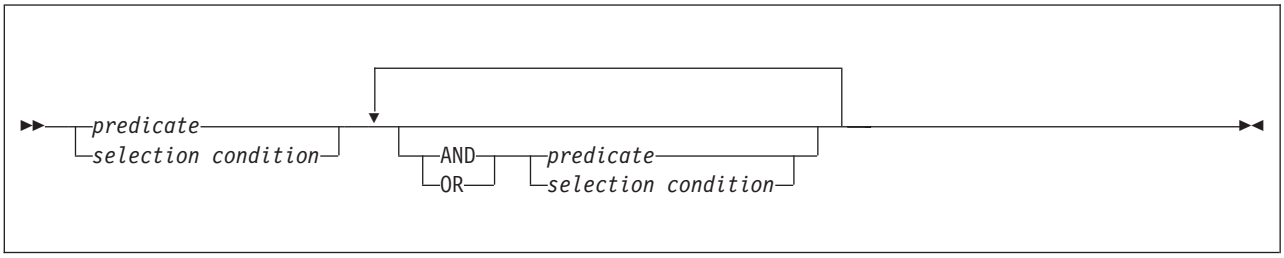
correlation-stats-spec:



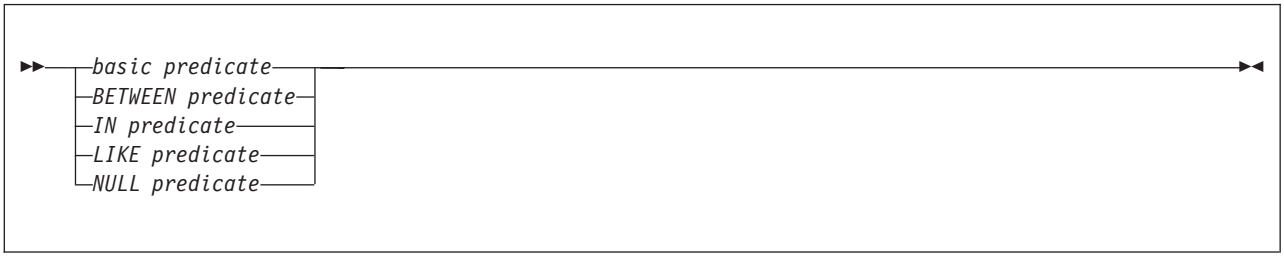
FROM-TABLE-spec:



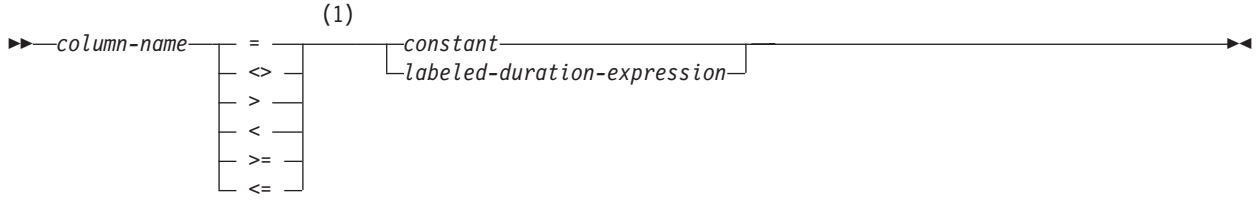
selection-condition-spec:



predicate:



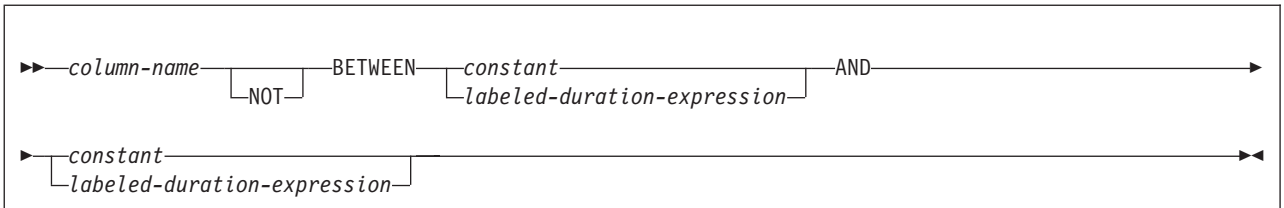
basic predicate:



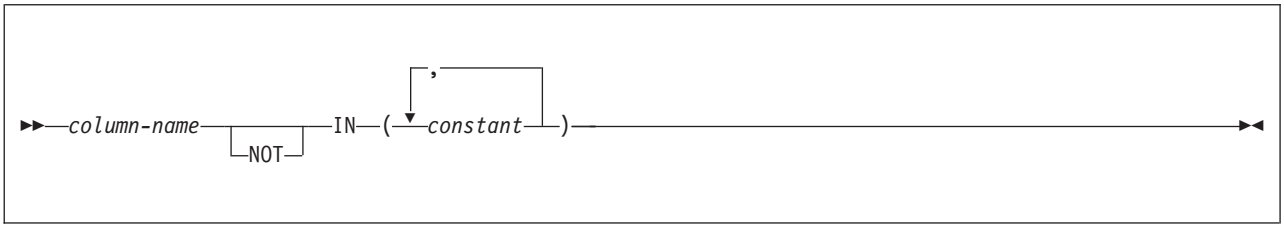
Notes:

1 The following forms of the comparison operators are also supported in basic and quantified predicates: !=, !<, and !>. For details, see "comparision operators" on page 439.

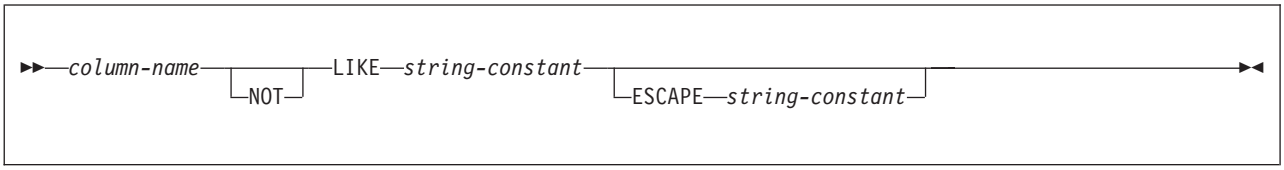
BETWEEN predicate:



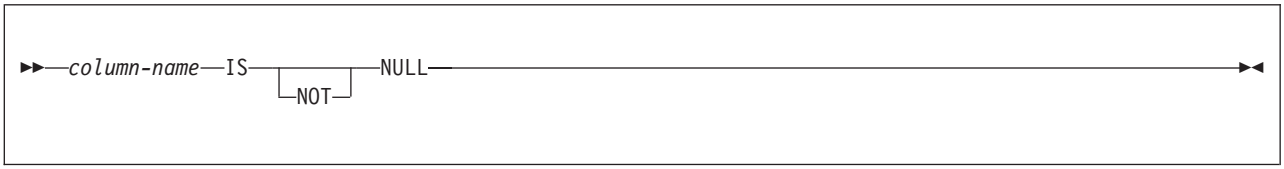
IN predicate:



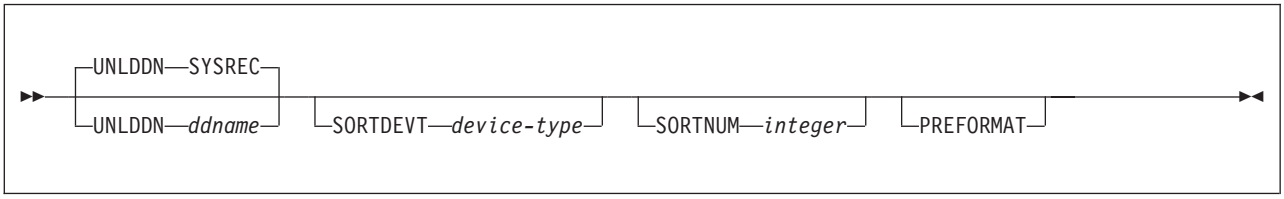
LIKE predicate:



NULL predicate:



reorg tablespace options:



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

TABLESPACE *database-name.table-space-name*
Specifies the table space (and, optionally, the database to which it belongs) that is to be reorganized.

If you reorganize a table space, its indexes are also reorganized.

database-name
Is the name of the database to which the table space belongs. The name cannot be DSNDB07. The **default** is **DSNDB04**.

table-space-name
Is the name of the table space that is to be reorganized. The name cannot be SYSUTILX if the specified database name is DSNDB01.

LIST *listdef-name*
Specifies the name of a previously defined LISTDEF list name. The utility allows one LIST keyword for each control statement of REORG TABLESPACE. The list must contain only table spaces.

Do not specify FROM TABLE, STATISTICS TABLE *table-name*, or STATISTICS INDEX *index-name* with REORG TABLESPACE LIST. If you want to collect inline statistics for a list of table spaces, specify STATISTICS TABLE (ALL). If you want to collect inline statistics for a list of indexes, specify STATISTICS INDEX (ALL). Do not specify PART with LIST.

REORG TABLESPACE is invoked once for each item in the list.

For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

REUSE
When used with SHRLEVEL NONE, specifies that REORG is to logically reset and reuse DB2-managed data sets without deleting and redefining them. If you do not specify REUSE and SHRLEVEL NONE, DB2 deletes and redefines DB2-managed data sets to reset them.

If a data set has multiple extents, the extents are not released if you use the REUSE parameter.

REUSE does not apply if you also specify SHRLEVEL REFERENCE or CHANGE.

SCOPE
Indicates the scope of the reorganization of the specified table space or of one or more specified partitions.

ALL
Indicates that you want the specified table space or one or more partitions to be reorganized. The **default** is **ALL**.

PENDING
Indicates that you want the specified table space or one or more partitions to be reorganized only if they are in REORG-pending (REORP or AREO*) status.

PART *integer*
PART *integer1*:*integer2*
Identifies a partition range that is to be reorganized. You can reorganize a single partition of a partitioned table space, or a range of partitions within a partitioned table space. *integer* must be in the range from 1 to the number of partitions that are defined for the table space or partitioning index. The maximum is 4096.

integer Designates a single partition.

integer1:*integer2*
Designates a range of existing table space partitions from *integer1* through *integer2*.*integer2* must be greater than *integer1*.

If you omit the PART keyword, the entire table space is reorganized.

If you specify the PART keyword for a LOB table space, DB2 issues an error message, and utility processing terminates with return code 8.

If you specify a partition range and the high or low partitions in the list are in a REORG-pending state, the adjacent partition that is outside the specified range must not be in REORG-pending state; otherwise, the utility terminates with an error.

REBALANCE
Specifies that REORG TABLESPACE is to set new partition boundaries so that rows are evenly distributed across the reorganized partitions. If the columns that are used in defining the partition boundaries have many duplicate values within the data rows, even balancing is not always possible. Specify REBALANCE for more than one partition; if you specify a single partition for rebalancing, REORG TABLESPACE ignores the specification.

You can specify REBALANCE with SHRLEVEL NONE or SHRLEVEL REFERENCE. REBALANCE cannot be specified with SHRLEVEL CHANGE or SCOPE PENDING. Also, do not specify REBALANCE for partitioned table spaces with LOB columns.

When you specify REBALANCE, you must create an inline copy by performing one of the following actions:

- Provide a SYSCOPY DD statement in the JCL.
- Use the TEMPLATE utility to dynamically allocate the SYSCOPY data set.
- Specify a DD name with the COPYDDN option in the REORG control statement and specify either a corresponding DD statement or TEMPLATE statement.

For more information about creating inline copies, see "Using inline copy with REORG TABLESPACE" on page 470.

| For additional restrictions, see "Restrictions when using REBALANCE" on page 450.

| At completion, DB2 invalidates plans, packages, and the dynamic cache.

LOG
Specifies whether records are to be logged during the RELOAD phase of REORG. If the records are not logged, the table space is recoverable only after an image copy is taken. If you specify COPYDDN, RECOVERYDDN, SHRLEVEL REFERENCE, or SHRLEVEL CHANGE, an image copy is taken during REORG execution.

YES
Specifies that log records are to be taken during the RELOAD phase. This option is not allowed for any table space in DSNDB01 or DSNDB06, or if the SHRLEVEL REFERENCE or CHANGE options are used.

If you specify SHRLEVEL NONE (explicitly or by default), the **default** is **YES**.

You must specify LOG YES (explicitly or by default) for a LOB table space. Logging occurs only if the LOB table space is defined with the LOG YES attribute. If the LOB table space is defined with the LOG NO attribute, the LOB table space is put in COPY-pending status after the REORG.

NO
Specifies that records are not to be logged. This option puts the table space in COPY-pending status if either of these conditions is true:

- REORG is executed at the local site, and COPYDDN, SHRLEVEL REFERENCE, and SHRLEVEL CHANGE options are not specified.
- REORG is executed at the remote site, and RECOVERYDDN is not specified.

SORTDATA

YES
Specifies that the data is to be unloaded by a table space scan, and sorted
in clustering order. The **default** is **SORTDATA YES** unless you specify
UNLOAD ONLY or UNLOAD EXTERNAL. If you specify one of these
options, the default is SORTDATA NO.

NO
Specifies that the data is to be unloaded in the order of the clustering
index. SORTDATA NO cannot be specified with SHRLEVEL CHANGE.

Specify SORTDATA NO if one of the following conditions is true:

- The data is in or near perfect clustering order, and the REORG utility is used to reclaim space from dropped tables.
- The data set is very large, and an insufficient amount of disk space is available for sorting.

| SORTDATA is ignored for some of the catalog and directory table spaces; see "Reorganizing the catalog and directory" on page 466.

NOSYSREC
Specifies that the output of sorting (if a clustering index exists) is the input to reloading, without the REORG TABLESPACE utility using an unload data set.
| You can specify this option only if the REORG TABLESPACE job includes

SHRLEVEL REFERENCE or SHRLEVEL NONE, and only if you do not specify UNLOAD PAUSE or UNLOAD ONLY. See "Omitting the output data set" on page 464 for additional information about using this option.

COPYDDN (*ddname1,ddname2***)**
Specifies the DD statements for the primary (*ddname1*) and backup (*ddname2*) copy data sets for the image copy.

ddname1 and *ddname2* are the DD names.

The **default** is **SYSCOPY** for the primary copy. A full image copy data set is created when REORG executes. This copy is called an inline copy. (For more information about inline copies, see "Using inline copy with REORG TABLESPACE" on page 470.) The name of the data set is listed as a row in the SYSIBM.SYSCOPY catalog table with ICTYPE='R' (as it is for the COPY SHRLEVEL REFERENCE option). The table space does not remain in COPY-pending status regardless of which LOG option you specify.

If you specify SHRLEVEL NONE (explicitly or by default) for REORG, and COPYDDN is not specified, an image copy is not created at the local site.

COPYDDN(SYSCOPY) is assumed, and a DD statement for SYSCOPY is required if either of the following conditions are true:

- You specify REORG SHRLEVEL REFERENCE or CHANGE, and you do not specify COPYDDN.
- A table space or partition is in REORG-pending (REORP) status.
- You specify REBALANCE.

The COPYDDN keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

REORG cannot take inline copies of LOB table spaces.

RECOVERYDDN *(ddname3,ddname4)*
Specifies the DD statements for the primary *(ddname3)* and backup *(ddname4)* copy data sets for the image copy at the recovery site.

ddname3 and *ddname4*are the DD names.

You cannot have duplicate image copy data sets. The same rules apply for RECOVERYDDN as for COPYDDN.

The RECOVERYDDN keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

An inline copy cannot be created for a LOB table space while running a REORG job on this table space.

SHRLEVEL
Specifies the method that is to be used for the reorganization. The parameter following SHRLEVEL indicates the type of access that is to be allowed during the RELOAD phase of REORG.

For a LOB table space, you must specify SHRLEVEL NONE (explicitly or by default).

NONE

Specifies that reorganization is to operate as follows:

- Unloading from the area that is being reorganized (while applications can read but cannot write to the area)
- Reloading into that area (while applications have no access), and then allowing read-write access again

The **default** is **NONE**.

If you specify NONE (explicitly or by default), you cannot specify the following parameters:

- MAPPINGTABLE
- MAXRO
- LONGLOG
- DELAY
- DEADLINE
- DRAIN_WAIT
- RETRY
- RETRY_DELAY

Restriction: If you specify UNLOAD PAUSE or UNLOAD ONLY, you cannot specify NOSYSREC.

REFERENCE

Specifies that reorganization is to operate as follows:

- Unloading from the area that is being reorganized (while applications can read but cannot write to the area)
- Reloading into a shadow copy of that area (while applications can read but cannot write to the original copy)
- Switching the future access of an application from the original copy to the shadow copy by exchanging the names of the data sets, and then allowing read-write access again

To determine which data sets are required when you execute REORG SHRLEVEL REFERENCE, see "Data sets that REORG TABLESPACE uses" on page 453.

If you specify REFERENCE, you cannot specify the following parameters:

- LOG. Reorganization with REFERENCE always creates an image copy and always refrains from logging records during reloading.
- UNLOAD. Reorganization with REFERENCE always performs UNLOAD CONTINUE.
- MAPPINGTABLE.
- MAXRO.
- LONGLOG.
- DELAY.

Restriction: You cannot use SHRLEVEL REFERENCE for a LOB table space.

CHANGE

Specifies that reorganization is to operate as follows:

- By unloading from the area that is being reorganized (while applications can read and write to the area)
- Reloading into a shadow copy of that area (while applications have read-write access to the original copy of the area)
- Applying the log of the original copy to the shadow copy (while applications can read and usually write to the original copy)
- Switching the future access of an application from the original copy to the shadow copy by exchanging the names of the data sets, and then allowing read-write access again

To determine which data sets are required when you execute REORG SHRLEVEL CHANGE, see "Data sets that REORG TABLESPACE uses" on page 453.

If you specify CHANGE, you cannot specify the following parameters:

- LOG. Reorganization with CHANGE always creates an image copy and always refrains from logging records during reloading.
- UNLOAD. Reorganization with CHANGE always performs UNLOAD CONTINUE.

If you specify CHANGE, you must create a mapping table and specify the name of the mapping table with the MAPPINGTABLE option.

Restriction: You cannot use SHRLEVEL CHANGE for a LOB table space or a catalog or directory table space with links.

DEADLINE

Specifies the deadline for the SWITCH phase to begin. If DB2 estimates that the SWITCH phase will not begin by the deadline, DB2 issues the messages that the DISPLAY UTILITY command would issue and then terminates the reorganization.

If REORG SHRLEVEL REFERENCE or SHRLEVEL CHANGE terminates because of a DEADLINE specification, DB2 issues message DSNU374I with reason code 2 but does not set a restrictive status.

NONE

Specifies that a deadline by which the SWITCH phase of log processing must begin does not exist. The **default** is **NONE**.

timestamp

Specifies the deadline for the SWITCH phase of log processing to begin. This deadline must not have already occurred when REORG is executed.

labeled-duration-expression

Calculates the deadline for the SWITCH phase of log processing to begin. The calculation is based on either CURRENT TIMESTAMP or CURRENT DATE. You can add or subtract one or more *constant* value to specify the deadline. This deadline must not have already occurred when REORG is executed. CURRENT TIMESTAMP and CURRENT DATE are evaluated once, when the REORG statement is first processed. If a list of objects is specified, the same value will be in effect for all objects in the list.

CURRENT_DATE

Specifies that the deadline is to be calculated based on the CURRENT DATE.

CURRENT_TIMESTAMP
Specifies that the deadline is to be calculated based on the CURRENT TIMESTAMP.

constant
Indicates a unit of time and is followed by one of the seven duration keywords: YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS, or MICROSECONDS. The singular form of these words is also acceptable: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, MICROSECOND.

DRAIN_WAIT *integer*
Specifies the number of seconds that the utility waits when draining the table space or index. The specified time is the aggregate time for objects that are to be reorganized. This value overrides the values that are specified by IRLMRWT and UTIMOUT. If the keyword is omitted or if a value of 0 is specified, the utility uses the IRLMRWT and UTIMOUT values for regular draining. Valid values for *integer* are from 0 to 1800.

RETRY *integer*
Specifies the maximum number of retries that REORG is to attempt. Valid values for *integer* are from 0 to 255. If the keyword is omitted, the utility does not attempt a retry.

Specifying RETRY can lead to increased processing costs and can result in multiple or extended periods of read-only access. For example, when you specify RETRY and SHRLEVEL CHANGE, the size of the copy that is taken by REORG might increase.

RETRY_DELAY *integer*
Specifies the minimum duration, in seconds, between retries. Valid values for *integer* are from 1 to 1800. The **default** is **300** seconds.

MAPPINGTABLE *table-name*
Specifies the name of the mapping table that REORG TABLESPACE is to use to map between the RIDs of data records in the original copy of the area and the corresponding RIDs in the shadow copy. This parameter is required if you specify SHRLEVEL CHANGE, and you must create a mapping table and an index for it before running REORG TABLESPACE. See "Before running REORG TABLESPACE" on page 449 for the columns and the index that the mapping table must include. Enclose the table name in quotation marks if the name contains a blank.

MAXRO *integer*
Specifies the maximum amount of time for the last iteration of log processing. During that iteration, applications have read-only access.

The actual execution time of the last iteration might exceed the specified value for MAXRO.

The ALTER UTILITY command can change the value of MAXRO.

The **default** is **300** seconds.

integer *integer* is the number of seconds. Specifying a small positive value reduces the length of the period of read-only access, but it might increase the elapsed time for REORG to complete. If you specify a huge positive value, the second iteration of log processing is probably the last iteration.

DEFER
Specifies that the iterations of log processing with read-write access can

continue indefinitely. REORG never begins the final iteration with read-only access, unless you change the MAXRO value with ALTER UTILITY.

If you specify DEFER, you should also specify LONGLOG CONTINUE.

If you specify DEFER, and DB2 determines that the actual time for an iteration and the estimated time for the next iteration are both less than 5 seconds, DB2 adds a 5 second pause to the next iteration. This pause reduces consumption of processor time. The first time this situation occurs for a given execution of REORG, DB2 sends message DSNU362I to the console. The message states that the number of log records that must be processed is small and that the pause occurs. To change the MAXRO value and thus cause REORG to finish, execute the ALTER UTILITY command. DB2 adds the pause whenever the situation occurs; however, DB2 sends the message only if 30 minutes have elapsed since the last message was sent for a given execution of REORG.

DRAIN
Specifies drain behavior at the end of the log phase after the MAXRO threshold is reached and when the last iteration of the log is to be applied.

WRITERS
Specifies the current default action, in which DB2 drains only the writers during the log phase after the MAXRO threshold is reached and subsequently issues DRAIN ALL on entering the switch phase.

ALL Specifies that DB2 is to drain all readers and writers during the log phase, after the MAXRO threshold is reached.

Consider specifying DRAIN ALL if the following conditions are both true:

- SQL update activity is high during the log phase.
- The default behavior results in a large number of -911 SQL error codes.

LONGLOG
Specifies the action that DB2 is to perform, after sending a message to the console, if the number of records that the next iteration of logging is to process is not sufficiently lower than the number that the previous iterations processed. This situation means that the reading of the log by the REORG TABLESPACE utility is not being done at the same time as the writing of the application log.

CONTINUE
Specifies that until the time on the JOB statement expires, DB2 is to continue performing reorganization, including iterations of log processing, if the estimated time to perform an iteration exceeds the time that is specified for MAXRO.

A value of DEFER for MAXRO and a value of CONTINUE for LONGLOG together mean that REORG is to continue allowing access to the original copy of the area that is being reorganized and does not switch to the shadow copy. The user can execute the ALTER UTILITY command with a large value for MAXRO to initiate switching.

The **default** is **CONTINUE.**

TERM Specifies that DB2 is to terminate the reorganization after the delay that is specified by the DELAY parameter.

DRAIN
Specifies that DB2 is to drain the write claim class after the delay that is specified by the DELAY parameter. This action forces the final iteration of log processing to occur.

DELAY *integer*
Specifies the minimum interval between the time that REORG sends the LONGLOG message to the console and the time that REORG performs the action that is specified by the LONGLOG parameter.

integer is the number of seconds. The **default** is **1200**.

TIMEOUT
Specifies the action that is to be taken if the REORG utility gets a time-out condition while trying to drain objects in either the log or switch phases.

ABEND
Indicates that, if a time-out condition occurs, DB2 is to leave the objects in a UTRO or UTUT state.

TERM
Indicates that DB2 is to behave as follows if you specify the TERM option and a time-out condition occurs:

1. DB2 issues an implicit TERM UTILITY command, causing the utility to end with a return code 8.
2. DB2 issues the DSNU590I and DSNU170I messages.
3. DB2 leaves the objects in a read-write state.

FASTSWITCH
Specifies which switch methodology is to be used for a given reorganization.

YES
Enables the SWITCH phase to use the FASTSWITCH methodology. This option is not allowed for the catalog (DSNDB06) or directory (DSNDB01).

The **default** is **YES**.

NO
Causes the SWITCH phase to use IDCAMS RENAME methodology.

OFFPOSLIMIT *integer*
Indicates that the specified value is to be compared to the value that DB2 calculates for the explicit clustering indexes of every table in the specified partitions that are in SYSIBM.SYSINDEXPART. The calculation is computed as follows:

```
(NEAROFFPOSF + FAROFFPOSF) × 100 / CARDF
```

Alternatively, DB2 checks the values in SYSINDEXPART for a single nonpartitioned table space, or for each partition if you specified an entire partitioned table space as the target object. If at least one calculated value exceeds the OFFPOSLIMIT value, REORG is performed or recommended. This option is valid for non-LOB table spaces only.

integer is the value that is to be compared and can range from 0 to 65535. The **default** value is **10**.

Note: If you specify OFFPOSLIMIT with REORG DISCARD to remove unwanted rows in a table space, OFFPOSLIMIT will override DISCARD. This may result in some rows not being removed.

INDREFLIMIT *integer*
Indicates that the specified value is to be compared to the value that DB2 calculates for the specified partitions in SYSIBM.SYSTABLEPART for the specified table space. The calculation is computed as follows:

```
(NEARINDREF + FARINDREF) × 100 / CARDF
```

Alternatively, DB2 checks the values in SYSTABLEPART for a single nonpartitioned table space, or for each partition if you specified an entire partitioned table space as the target object. If at least one calculated value exceeds the calculated value exceeds the INDREFLIMIT value, REORG is performed or recommended. This option is valid for non-LOB table spaces only.

integer is the value that is to be compared and can range from 0 to 65535. The **default** value is **10**.

REPORTONLY
Specifies that REORG is only to be recommended, not performed. REORG produces a report with one of the following return codes:

1 No limit met; no REORG is to be performed or recommended.
2 REORG is to be performed or recommended.

UNLOAD
Specifies whether the utility job is to continue processing or end after the data is unloaded. Unless you specify UNLOAD EXTERNAL, data can be reloaded only into the same table and table space (as defined in the DB2 catalog) on the same subsystem. (This does not preclude VSAM redefinition during UNLOAD PAUSE.)

You must specify UNLOAD ONLY for the data set to be in a format that is compatible with the FORMAT UNLOAD option of LOAD. However, with LOAD, you can load the data only into the same object from which it is unloaded.

This option is valid for non-LOB table spaces only.

You must specify UNLOAD EXTERNAL for the data set to be in a format that is usable by LOAD without the FORMAT UNLOAD option. With UNLOAD EXTERNAL, you can load the data into any table with compatible columns in any table space on any DB2 subsystem.

CONTINUE
Specifies that, after the data has been unloaded, the utility is to continue processing. An edit routine can be called to decode a previously encoded data row if an index key requires extraction from that row.

If you specify DISCARD, rows are decompressed and edit routines are decoded. If you also specify DISCARD to a file, rows are decoded by field procedure, and the following columns are converted to DB2 external format:

- SMALLINT
- INTEGER
- FLOAT
- DECIMAL
- TIME
- TIMESTAMP

Otherwise, edit routines or field procedures are bypassed on both the UNLOAD and RELOAD phases for table spaces. Validation procedures are not invoked during either phase.

The **default** is **CONTINUE**.

PAUSE
Specifies that, after the data has been unloaded, processing is to end. The utility stops and the RELOAD status is stored in SYSIBM.SYSUTIL so that processing can be restarted with RELOAD RESTART(PHASE).

This option is useful if you want to redefine data sets during reorganization. For example, with a user-defined data set, you can:
- Run REORG with the UNLOAD PAUSE option.
- Redefine the data set by using Access Method Services.
- Restart REORG by resubmitting the previous job and specifying RESTART(PHASE).

However, you cannot use UNLOAD PAUSE if you specify the LIST option.

ONLY
Specifies that, after the data has been unloaded, the utility job ends and the status that corresponds to this utility ID is removed from SYSIBM.SYSUTIL.

If you specify UNLOAD ONLY with REORG TABLESPACE, any edit routine or field procedure is executed during record retrieval in the unload phase.

This option is not allowed for any table space in DSNDB01 or DSNDB06.

The DISCARD and WHEN options are not allowed with UNLOAD ONLY.

EXTERNAL
Specifies that, after the data has been unloaded, the utility job is to end and the status that corresponds to this utility ID is removed.

The UNLOAD utility has more functions. If you specify UNLOAD EXTERNAL with REORG TABLESPACE, rows are decompressed, edit routines are decoded, field procedures are decoded, and SMALLINT, INTEGER, FLOAT, DECIMAL, DATE, TIME, and TIMESTAMP columns are converted to DB2 external format. Validation procedures are not invoked.

This option is not allowed for any table space in DSNDB01 or DSNDB06.

The DISCARD option is not allowed with UNLOAD EXTERNAL.

NOPAD
Specifies that the variable-length columns in the unloaded or discarded records are to occupy the actual data length without additional padding. The unloaded records can have varying lengths. If you do not specify NOPAD, default REORG processing pads variable-length columns in the unloaded or discarded records to their maximum length; the unloaded or discarded records have equal lengths for each table.

You can specify the NOPAD option only with UNLOAD EXTERNAL or with UNLOAD DISCARD.

Although the LOAD utility processes records with variable-length columns that were unloaded or discarded with the NOPAD option, these records cannot be processed by applications that process only fields that are in fixed positions.

For the generated LOAD statement to provide a NULLIF condition for fields that are not in a fixed position, DB2 generates an input field definition with a name in the form of DSN_NULL_IND_*nnnnn*, where *nnnnn* is the number of the associated column.

For example, the LOAD statement that is generated for the EMP sample table looks similar to the LOAD statement that is in Figure 77.

```
LOAD DATA INDDN SYSREC   LOG NO  RESUME YES
 EBCDIC CCSID(00500,00000,00000)
 INTO TABLE "DSN8810 "."EMP              "
 WHEN(00004:00005 = X'0012')
 ( "EMPNO             " POSITION(00007:00012) CHAR(006)
 , "FIRSTNME          " POSITION(00013)       VARCHAR
 , "MIDINIT           " POSITION(*)           CHAR(001)
 , "LASTNAME          " POSITION(*)           VARCHAR
 ,  DSN_NULL_IND_00005  POSITION(*)           CHAR(1)
 , "WORKDEPT          " POSITION(*)           CHAR(003)
                         NULLIF(DSN_NULL_IND_00005)=X'FF'
 ,  DSN_NULL_IND_00006  POSITION(*)           CHAR(1)
 , "PHONENO           " POSITION(*)           CHAR(004)
                         NULLIF(DSN_NULL_IND_00006)=X'FF'
 ,  DSN_NULL_IND_00007  POSITION(*)           CHAR(1)
 , "HIREDATE          " POSITION(*)           DATE EXTERNAL
                         NULLIF(DSN_NULL_IND_00007)=X'FF'
 ,  DSN_NULL_IND_00008  POSITION(*)           CHAR(1)
 , "JOB               " POSITION(*)           CHAR(008)
                         NULLIF(DSN_NULL_IND_00008)=X'FF'
 ,  DSN_NULL_IND_00009  POSITION(*)           CHAR(1)
 , "EDLEVEL           " POSITION(*)           SMALLINT
                         NULLIF(DSN_NULL_IND_00009)=X'FF'
 ,  DSN_NULL_IND_00010  POSITION(*)           CHAR(1)
 , "SEX               " POSITION(*)           CHAR(001)
                         NULLIF(DSN_NULL_IND_00010)=X'FF'
 ,  DSN_NULL_IND_00011  POSITION(*)           CHAR(1)
 , "BIRTHDATE         " POSITION(*)           DATE EXTERNAL
                         NULLIF(DSN_NULL_IND_00011)=X'FF'
 ,  DSN_NULL_IND_00012  POSITION(*)           CHAR(1)
 , "SALARY            " POSITION(*)           DECIMAL
                         NULLIF(DSN_NULL_IND_00012)=X'FF'
 ,  DSN_NULL_IND_00013  POSITION(*)           CHAR(1)
 , "BONUS             " POSITION(*)           DECIMAL
                         NULLIF(DSN_NULL_IND_00013)=X'FF'
 ,  DSN_NULL_IND_00014  POSITION(*)           CHAR(1)
 , "COMM              " POSITION(*)           DECIMAL
                         NULLIF(DSN_NULL_IND_00014)=X'FF'
 )
```

Figure 77. Sample LOAD statement generated by REORG TABLESPACE with the NOPAD keyword

FROM TABLE

Specifies the tables that are to be reorganized. The table space that is specified in REORG TABLESPACE can store more than one table. All tables that are specified by FROM TABLE statements must be unique. All tables are unloaded for UNLOAD EXTERNAL, and all tables might be subject to DISCARD. If you specify UNLOAD EXTERNAL and want to limit which tables and rows are unloaded, specify FROM TABLE with the WHEN option. If you specify DISCARD, you must qualify the rows that you want to discard by specifying FROM TABLE with the WHEN option.

Do not specify FROM TABLE with REORG TABLESPACE LIST.

table-name

Specifies the name of the table that is to be qualified by the following WHEN clause. The table must be described in the catalog and must not be a catalog table. If the table name is not qualified by an authorization ID,

the authorization ID of the person who invokes the utility job step is used as the qualifier of the table name. Enclose the table name in quotation marks if the name contains a blank.

WHEN

Indicates which records in the table space are to be unloaded (for UNLOAD EXTERNAL) or discarded (for DISCARD). If you do not specify a WHEN clause for a table in the table space, all of the records are unloaded (for UNLOAD EXTERNAL), or none of the records is discarded (for DISCARD).

The option following WHEN describes the conditions for UNLOAD or DISCARD of records from a table and must be enclosed in parentheses.

selection condition

Specifies a condition that is true, false, or unknown about a specific row. When the condition is true, the row qualifies for UNLOAD or DISCARD. When the condition is false or unknown, the row does not qualify.

A selection condition consists of at least one predicate and any *logical operators* (AND, OR, NOT). The result of a selection condition is derived by applying the specified *logical operators* to the result of each specified predicate. If logical operators are not specified, the result of the selection condition is the result of the specified predicate.

Selection conditions within parentheses are evaluated first. If the order of evaluation is not specified by parentheses, AND is applied before OR.

If the control statement is in the same encoding scheme as the input data, you can code character constants in the control statement. Otherwise, if the control statement is not in the same encoding scheme as the input data, you must code the condition with hexadecimal constants. If the target table is ASCII, any character constants must be specified in hexadecimal. For example, if the table space is in EBCDIC and the control statement is in UTF-8, use (1:1)=X'F1' in the condition rather than (1:1)='1'.

| **Restriction:** REORG TABLESPACE cannot filter rows that contain encrypted data.

predicate

A *predicate* specifies a condition that is true, false, or unknown about a given row or group.

basic predicate

Specifies the comparison of a column with a constant. If the value of the column is null, the result of the predicate is unknown. Otherwise, the result of the predicate is true or false.

Predicate	Is true if and only if
column-name **= constant**	The column is equal to the constant or labeled duration expression.
column-name **< > constant**	The column is not equal to the constant or labeled duration expression.
column-name **> constant**	The column is greater than the constant or labeled duration expression.
column-name **< constant**	The column is less than the constant or labeled duration expression.

column-name **> = constant** The column is greater than or equal to the constant or labeled duration expression.

column-name **< = constant** The column is less than or equal to the constant or labeled duration expression.

Comparison operators: The following forms of the comparison operators are also supported in basic and quantified predicates: !=, !<, and !>, where ! means not. In addition, in code pages 437, 819, and 850, the forms ¬=, ¬<, and ¬> are supported. All these product-specific forms of the comparison operators are intended only to support existing REORG statements that use these operators and are not recommended for use in new REORG statements.

A not sign (¬), or the character that must be used in its place in certain countries, can cause parsing errors in statements that are passed from one DBMS to another. The problem occurs if the statement undergoes character conversion with certain combinations of source and target CCSIDs. To avoid this problem, substitute an equivalent operator for any operator that includes a not sign. For example, substitute '< >' for '¬=', '<=' for '¬>', and '>=' for '¬<'.

BETWEEN predicate

Indicates whether a given value lies between two other given values that are specified in ascending order. Each of the predicate's two forms (BETWEEN and NOT BETWEEN) has an equivalent search condition, as shown in Table 71. If relevant, the table also shows any equivalent predicates.

Table 71. BETWEEN predicates and their equivalent search conditions

Predicate	Equivalent predicate	Equivalent search condition
column BETWEEN *value1* AND *value2*	None	(*column* >= *value1* AND *column* <= *value2*)
column NOT BETWEEN *value1* AND *value2*	NOT(*column* BETWEEN *value1* AND *value2*)	(*column* < *value1* OR *column* > *value2*)

Note: The values can be constants or labeled duration expressions.

For example, the following predicate is true for any row when salary is greater than or equal to 10 000 and less than or equal to 20 000:

```
SALARY BETWEEN 10000 AND 20000
```

labeled-duration-expression

Specifies an expression that begins with the following special register values:

- CURRENT DATE (CURRENT_DATE is acceptable.)
- CURRENT TIMESTAMP (CURRENT_TIMESTAMP is acceptable.)

Optionally, the expression contains the arithmetic operations of addition or subtraction, expressed by a number followed by one of the seven duration keywords:

- YEARS (or YEAR)
- MONTHS (or MONTH)
- DAYS (or DAY)
- HOURS (or HOUR)
- MINUTES (or MINUTE)
- SECONDS (or SECOND)

- MICROSECONDS (or MICROSECOND)

Utilities evaluate a *labeled-duration-expression* as a timestamp and implicitly perform a conversion to a date if the comparison is with a date column.

Incrementing and decrementing CURRENT DATE: The result of adding a duration to a date, or of subtracting a duration from a date, is itself a date. (For the purposes of this operation, a month denotes the equivalent of a calendar page. Adding months to a date, then, is like turning the pages of a calendar, starting with the page on which the date appears.) The result must fall between the dates January 1, 0001 and December 31, 9999 inclusive.

Table 72 describes the effects of adding and subtracting years, months, days, and other dates.

Table 72. Effects of adding durations to and subtracting durations from CURRENT DATE

Value that is added or subtracted	Effect
Years	Adding or subtracting a duration of years affects only the year portion of the date. The month is unchanged, as is the day unless the result would be February 29 of a non-leap-year. In this case, the day portion of the result is set to 28.
Months	Adding or subtracting a duration of months affects only months and, if necessary, years. The day portion of the date is unchanged unless that day does not exist in the resulting month. (September 31, for example). In this case the day is set to the last day of the month. Adding a month to a date gives the same day one month later unless that day does not exist in the later month. In that case, the day in the result is set to the last day of the later month. For example, January 28 plus one month gives February 28; one month added to January 29, 30, or 31 results in either February 28 or, for a leap year, February 29. If one or more months is added to a given date and then the same number of months is subtracted from the result, the final date is not necessarily the same as the original date.
Days	Adding or subtracting a duration of days affects the day portion of the date, and potentially the month and year.
Dates	When a positive date duration is added to a date, or a negative date duration is subtracted from a date, the date is incremented by the specified number of years, months, and days. When a positive date duration is subtracted from a date, or a negative date duration is added to a date, the date is decremented by the specified number of days, months, and years.

The order in which labeled date durations are added to and subtracted from dates can affect the results. When you add labeled date durations to a date, specify them in the order of YEARS + MONTHS + DAYS. When you subtract labeled date durations from a date, specify them in the order of DAYS - MONTHS - YEARS. For example, to add one year and one day to a date, specify the following code:

```
CURRENT DATE + 1 YEAR + 1 DAY
```

To subtract one year, one month, and one day from a date, specify the following code:

```
CURRENT DATE - 1 DAY - 1 MONTH - 1 YEAR
```

Incrementing and decrementing timestamps: The result of adding a duration to a timestamp, or of subtracting a duration from a timestamp, is itself a timestamp. Date and time arithmetic is performed as previously defined, except that an overflow or underflow of hours is carried into the date part of the result, which must be within the range of valid dates. For example, if the current date is January 15 and the current time is 20:00, `CURRENT_TIMESTAMP+8 HOURS` yields January 16, 04:00. Likewise, `CURRENT_TIMESTAMP-22 HOURS` yields January 14, 22:00.

IN predicate
Specifies that a value is to be compared with a set of values. In the IN predicate, the second operand is a set of one or more values that are specified by constants. Each of the predicate's two forms (IN and NOT IN) has an equivalent search condition, as shown in Table 73.

Table 73. IN predicates and their equivalent search conditions

Predicate	Equivalent search condition
value1 IN (*value1*, *value2*,..., *valuen*)	(*value1* = *value2* OR ... OR *value1* = *valuen*)
value1 NOT IN (*value1*, *value2*,..., *valuen*)	*value1* ¬= *value2* AND ... AND *value1* ¬= *valuen*)

Note: The values can be constants or labeled duration expressions.

For example, the following predicate is true for any row with an employee in department D11, B01, or C01:

```
WORKDEPT IN ('D11', 'B01', 'C01')
```

LIKE predicate
Qualifies strings that have a certain pattern. Specify the pattern by using a string in which the underscore and percent sign characters can be used as wildcard characters. The underscore character (_) represents a single, arbitrary character. The percent sign (%) represents a string of zero or more arbitrary characters.

In this description, let *x* denote the column that is to be tested and *y* denote the pattern in the string constant.

The following rules apply to predicates of the form "*x* LIKE *y*...". If NOT is specified, the result is reversed.

- When *x* or *y* is null, the result of the predicate is unknown.
- When *y* is empty and *x* is not empty, the result of the predicate is false.
- When *x* is empty and *y* is not empty, the result of the predicate is false unless *y* consists only of one or more percent signs.
- When *x* and *y* are both empty, the result of the predicate is true.
- When *x* and *y* are both not null, the result of the predicate is true if *x* matches the pattern in *y* and false if *x* does not match the pattern in *y*.

The pattern string and the string that is to be tested must be of the same type; that is, both *x* and *y* must be character strings, or both *x*

and *y* must be graphic strings. When *x* and *y* are graphic strings, a character is a DBCS character. When *x* and *y* are character strings and *x* is not mixed data, a character is an SBCS character, and *y* is interpreted as SBCS data regardless of is subtype. The rules for mixed-data patterns are described in "Strings and patterns" on page 443.

Within the pattern, a percent sign (%) or underscore character (_) can represent the literal occurrence of a percent sign or underscore character. To have a literal meaning, each character must be preceded by an escape character.

The ESCAPE clause designates a single character. You can use that character, and only that character, multiple times within the pattern as an escape character. When the ESCAPE clause is omitted, no character serves as an escape character and percent signs and underscores in the pattern can only be used to represent arbitrary characters; they cannot represent their literal occurrences.

The following rules apply to the use of the ESCAPE clause:

- The ESCAPE clause cannot be used if *x* is mixed data.
- If *x* is a character string, the data type of the string constant must be character string. If *x* is a graphic string, the data type of the string constant must be graphic string. In both cases, the length of the string constant must be 1.
- The pattern must not contain the escape character except when followed by the escape character, '%', or '_'. For example, if '+' is the escape character, any occurrences of '+' other than '++', '+_', or '+%' in the pattern is an error.

When that pattern does not include escape characters, a simple description of its meaning is:

- The underscore character (_) represents a single, arbitrary character.
- The percent sign (%) represents a string of zero or more arbitrary characters.
- Any other character represents a single occurrence of itself.

Strings and patterns

The string *y* is interpreted as a sequence of the minimum number of substring specifiers, such that each character of *y* is part of exactly one substring specifier. A substring specifier is an underscore, a percent sign, or any non-empty sequence of characters other than an underscore or percent sign.

The string *x* matches the pattern *y* if a partitioning of *x* into substrings exists, such that:

- A substring of *x* is a sequence of zero or more contiguous characters, and each character of *x* is part of exactly one substring.
- If the *n*th substring specifier is an underscore, the *n*th substring of *x* is any single character.
- If the *n*th substring specifier is a percent sign, the *n*th substring of *x* is any sequence of zero or more characters.
- If the *n*th substring specifier is neither an underscore nor a percent sign, the *n*th substring of *x* is equal to that substring specifier and has the same length as that substring specifier.
- The number of substrings of *x* is the same as the number of substring specifiers.

When escape characters are present in the pattern string, an underscore, percent sign, or escape character represents a single occurrence of itself if and only if it is preceded by an odd number of successive escape characters.

Mixed-data patterns: If *x* is mixed data, the pattern is assumed to be mixed data, and its special characters are interpreted as follows:

- A single-byte underscore refers to one single-byte character; a double-byte underscore refers to one double-byte character.
- A percent sign, either single-byte or double-byte, refers to any number of characters of any type, either single-byte or double-byte.
- Redundant shift bytes in *x* or *y* are ignored.

NULL predicate

Specifies a test for null values.

If the value of the column is null, the result is true. If the value is not null, the result is false. If NOT is specified, the result is reversed.

KEEPDICTIONARY

Prevents REORG TABLESPACE from building a new compression dictionary when unloading the rows. The efficiency of REORG increases with the KEEPDICTIONARY option for the following reasons:

- The processing cost of building the compression dictionary is eliminated.
- Existing compressed rows do not need to be compressed again.
- Existing compressed rows do not need to be expanded, unless indexes require it or SORTDATA is used.

Possible reasons for not specifying KEEPDICTIONARY are:

- If the data has changed significantly since the last dictionary was built, rebuilding the dictionary might save a significant amount of space.
- If the current dictionary was built using the LOAD utility, building it using REORG might produce a better compression dictionary.

Note: You must use KEEPDICTIONARY to ensure that the compression
dictionary is maintained.

For more information about specifying or omitting the KEEPDICTIONARY option, see "Compressing data" on page 249.

KEEPDICTIONARY is valid only if a compression dictionary exists and the table space or partition that is being reorganized has the COMPRESS YES attribute. If a dictionary does not exist, one is built, a warning message is issued, and all the records are compressed.

Messages DSNU234I and DSNU244I, which show compression statistics, are not issued when you specify REORG UNLOAD CONTINUE KEEPDICTIONARY or REORG UNLOAD PAUSE KEEPDICTIONARY.

REORG ignores the KEEPDICTIONARY option if a partition that is being reorganized is in REORG-pending status.

| For information about data compression, see Part 5 (Volume 2) of *DB2*
| *Administration Guide*.

STATISTICS

Specifies that statistics for the table space or associated index, or both, are to be gathered; the statistics are reported or stored in the DB2 catalog. If statistics are collected with the default options, only the statistics for the table space are updated.

If you specify a table space partition or a range of partitions along with the STATISTICS keyword, DB2 collects statistics only for the specified table space partitions. This option is valid for non-LOB table spaces only.

You cannot collect inline statistics for indexes on specific catalog and directory tables. See "Reorganizing the catalog and directory" on page 466 for the list of unsupported catalog and directory tables.

Restriction: If you specify STATISTICS for encrypted data, DB2 might not
provide useful statistics on this data. If the utility is terminated with the
-TERM UTIL command after the STATISTICS have been updated in the
catalog, the statistics are not rolled back. A subsequent RUNSTATS utility may
be needed.

TABLE

Specifies the table for which column information is to be gathered. All tables must belong to the table space that is specified in the TABLESPACE option.

Do not specify STATISTICS TABLE *table-name* with REORG TABLESPACE LIST. Instead, specify STATISTICS TABLE (ALL).

(ALL)

Specifies that information is to be gathered for all columns of all tables in the table space.

(*table-name***)**

Specifies the tables for which column information is to be gathered. If you omit the qualifier, the user identifier for the utility job is used. Enclose the table name in quotation marks if the name contains a blank.

If you specify more than one table, you must repeat the TABLE option. Multiple TABLE options must be specified entirely before or after any INDEX keyword that may also be specified. For example, the INDEX keyword may not be specified between any two TABLE keywords.

SAMPLE *integer*
Indicates the percentage of rows to be sampled when collecting non-indexed column statistics. You can specify any value from 1 through 100. The **default** is **25**. The SAMPLE option is not allowed for LOB table spaces.

COLUMN
Specifies columns for which column information is to be gathered.

You can specify this option only if you specify a particular table for which statistics are to be gathered (TABLE (*table-name*)). If you specify particular tables and do not specify the COLUMN option, the default, COLUMN(ALL), is used. If you do not specify a particular table when using the TABLE option, you cannot specify the COLUMN option; however, COLUMN(ALL) is assumed.

(ALL)
Specifies that statistics are to be gathered for all columns in the table.

(column-name, ...)
Specifies the columns for which statistics are to be gathered.

You can specify a list of column names; the maximum is 10. If you specify more than one column, separate each name with a comma.

INDEX
Specifies indexes for which information is to be gathered. Column information is gathered for the first column of the index. All the indexes must be associated with the same table space, which must be the table space that is specified in the TABLESPACE option.

Do not specify STATISTICS INDEX *index-name* with REORG TABLESPACE LIST. Instead, specify STATISTICS INDEX (ALL).

(ALL) Specifies that the column information is to be gathered for all indexes that are defined on tables that are contained in the table space.

(*index-name***)**
Specifies the indexes for which information is to be gathered. Enclose the index name in quotation marks if the name contains a blank.

KEYCARD
Indicates that all of the distinct values in all of the 1 to *n* key column combinations for the specified indexes are to be collected. *n* is the number of columns in the index.

FREQVAL
Specifies that frequent-value statistics are to be collected. If you specify FREQVAL, you must also specify NUMCOLS and COUNT.

NUMCOLS
Indicates the number of key columns to concatenate together when you collect frequent values from the specified index. Specifying 3 means that DB2 is to collect frequent values on the concatenation of the first three key columns. The **default** is **1**, which means DB2 is to collect frequent values on the first key column of the index.

COUNT
Indicates the number of frequent values that are to be collected. For

example, specifying 15 means that DB2 is to collect 15 frequent values from the specified key columns. The **default** is **10**.

REPORT
Specifies whether a set of messages is to be generated to report the collected statistics.

NO
Indicates that the set of messages is not to be sent as output to SYSPRINT. The **default** is **NO**.

YES
Indicates that the set of messages is to be sent as output to SYSPRINT. The generated messages are dependent on the combination of keywords (such as TABLESPACE, INDEX, TABLE, and COLUMN) that are specified with the RUNSTATS utility. However, these messages are **not** dependent on the specification of the UPDATE option. REPORT YES always generates a report of SPACE and ACCESSPATH statistics.

UPDATE
Indicates whether the collected statistics are to be inserted into the catalog tables. UPDATE also allows you to select statistics that are used for access path selection or statistics that are used by database administrators.

ALL Indicates that all collected statistics are to be updated in the catalog. The **default** is **ALL**.

ACCESSPATH
Indicates that only the catalog table columns that provide statistics that are used for access path selection are to be updated.

SPACE
Indicates that only the catalog table columns that provide statistics to help database administrators assess the status of a particular table space or index are to be updated.

NONE
Indicates that no catalog tables are to be updated with the collected statistics. This option is valid only when REPORT YES is specified.

HISTORY
Specifies that all catalog table inserts or updates to the catalog history tables are to be recorded.

The default value is whatever value is specified in the STATISTICS HISTORY field on panel DSNTIPO.

ALL Indicates that all collected statistics are to be updated in the catalog history tables.

ACCESSPATH
Indicates that only the catalog history table columns that provide statistics that are used for access path selection are to be updated.

SPACE
Indicates that only space-related catalog statistics are to be updated in catalog history tables.

NONE
Indicates that no catalog history tables are to be updated with the collected statistics.

FORCEROLLUP
Specifies whether aggregation or rollup of statistics is to take place when RUNSTATS is executed even if statistics have not been gathered on some partitions; for example, partitions have not had any data loaded. Aggregate statistics are used by the optimizer to select the best access path.

YES Indicates that forced aggregation or rollup processing is to be done, even though some partitions might not contain data.

NO Indicates that aggregation or rollup is to be done only if data is available for all partitions.

If data is not available for all partitions, DSNU623I message is issued if the installation value for STATISTICS ROLLUP on panel DSNTIPO is set to NO.

PUNCHDDN *ddname*
Specifies the DD statement for a data set that is to receive the LOAD utility control statements that are generated by REORG TABLESPACE UNLOAD EXTERNAL or REORG TABLESPACE DISCARD FROM TABLE ... WHEN.

ddname is the DD name.

The **default** is **SYSPUNCH**.

PUNCHDDN is required if the limit key of the last partition of a partitioned table space has been reduced.

The PUNCHDDN keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

DISCARDDN *ddname*
Specifies the DD statement for a discard data set, which contains copies of records that meet the DISCARD FROM TABLE ... WHEN specification.

ddname is the DD name.

If you omit the DISCARDDN option, the utility saves discarded records only if a SYSDISC DD statement is in the JCL input.

The **default** is **SYSDISC**.

The **DISCARDDN** keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

UNLDDN *ddname*
Specifies the name of the unload data set.

ddname is the DD name of the unload data set. The **default** is **SYSREC**.

The UNLDDN keyword specifies either a DD name or a TEMPLATE name specification from a previous TEMPLATE control statement. If utility processing detects that the specified name is both a DD name in the current job step and a TEMPLATE name, the utility uses the DD name. For more information about TEMPLATE specifications, see Chapter 31, "TEMPLATE," on page 593.

SORTDEVT *device-type*
Specifies the device type for temporary data sets that are to be dynamically allocated by DFSORT.

device-type is the device type; it can be any device that is acceptable to the DYNALLOC parameter of the SORT or OPTION control statement for DFSORT.

SORTDEVT is ignored for the catalog and directory table spaces that are listed in "Reorganizing the catalog and directory" on page 466.

The utility does not allow a TEMPLATE specification to dynamically allocate sort work data sets. The SORTDEVT keyword controls dynamic allocation of these data sets.

SORTNUM *integer*
Specifies the number of temporary data sets that are to be dynamically allocated for all sorts that REORG performs.

integer is the number of temporary data sets that can range from 2 to 255.

If you omit SORTDEVT, SORTNUM is ignored. If you use SORTDEVT and omit SORTNUM, no value is passed to DFSORT. DFSORT uses its own default.

SORTNUM is ignored for the catalog and directory table spaces listed in "Reorganizing the catalog and directory" on page 466.

You need at least two sort work data sets for each sort. The SORTNUM value
applies to each sort invocation in the utility. For example, if there are three
indexes, SORTKEYS is specified, there are no constraints limiting parallelism,
and SORTNUM is specified as 8, then a total of 24 sort work data sets will be
allocated for a job.

Each sort work data set consumes both above the line and below the link
virtual storage, so if you specify too high a value for SORTNUM, the utility
may decrease the degree of parallelism due to virtual storage constraints, and
possibly decreasing the degree down to one, meaning no parallelism.

Important: The SORTNUM keyword will not be considered if ZPARM
UTSORTAL is set to YES and IGNSORTN is set to YES.

PREFORMAT
Specifies that the remaining pages are to be preformatted up to the high RBA in the table space and index spaces that are associated with the table that is specified in FROM TABLE *table-name* option. The preformatting occurs after the data is loaded and the indexes are built.

PREFORMAT can operate on an entire table space and its index spaces, or on a partition of a partitioned table space and its corresponding partitioning index space.

PREFORMAT is ignored if you specify UNLOAD ONLY or UNLOAD EXTERNAL.

For more information about the PREFORMAT option, see "Improving performance with LOAD or REORG PREFORMAT" on page 254.

DISCARD
Specifies that records that meet the specified WHEN conditions are to be discarded during REORG TABLESPACE UNLOAD CONTINUE or UNLOAD PAUSE. If you specify DISCARDDN or a SYSDISC DD statement in the JCL, discarded records are saved in the associated data set.

You can specify any SHRLEVEL option with DISCARD; however, if you specify SHRLEVEL CHANGE, modifications that are made during the reorganization to data rows that match the discard criteria are not permitted. In this case, REORG TABLESPACE terminates with an error.

If you specify DISCARD, rows are decompressed and edit routines are decoded. If you also specify DISCARD to a file, rows are decoded by field procedure, and the following columns are converted to DB2 external format:
- SMALLINT
- INTEGER
- FLOAT
- DECIMAL
- TIME
- TIMESTAMP

Otherwise, edit routines or field procedures are bypassed on both the UNLOAD and RELOAD phases for table spaces. Validation procedures are not invoked during either phase.

Do not specify DISCARD with the UNLOAD EXTERNAL or UNLOAD ONLY option.

Instructions for running REORG TABLESPACE

To run REORG TABLESPACE, you must:

1. Read "Before running REORG TABLESPACE" in this section.
2. Prepare the necessary data sets, as described in "Data sets that REORG TABLESPACE uses" on page 453.
3. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for REORG TABLESPACE, see "Sample REORG TABLESPACE control statements" on page 486.)
4. Prepare a utility control statement that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks" on page 461.
5. Check the compatibility table in "Concurrency and compatibility for REORG TABLESPACE" on page 480 if you want to run other jobs concurrently on the same target objects.
6. Plan for restart if the REORG TABLESPACE job doesn't complete, as described in "Terminating or restarting REORG TABLESPACE" on page 476.
7. Run REORG TABLESPACE by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Before running REORG TABLESPACE

Catalog and directory table spaces: Before running REORG on a catalog or directory table space, you must take an image copy. Be aware that for the DSNDB06.SYSCOPY catalog table space, and the DSNDB01.DBD01, and DSNDB01.SYSUTILX directory table spaces, REORG scans logs to verify that an image copy is available. If the scan of the logs does not find an image copy, DB2 requests archive logs.

Region size: The recommended minimum region size is 4096 KB. Region sizes greater than 32 MB enable increased parallelism for index builds.

Mapping table and SHRLEVEL CHANGE: Before running REORG TABLESPACE with SHRLEVEL CHANGE, you must create a mapping table and index for it. The table space that contains the mapping table must be segmented and cannot be the table space to be reorganized. To create the mapping table, use a CREATE TABLESPACE statement similar to the following statement:

```
CREATE TABLESPACE table-space-name SEGSIZE integer
```

The number of keys in the mapping index should not exceed 110% of the number of rows in the table space or partition that is to be reorganized. The mapping table must have only the columns and the index that are created by the following SQL statements:

```
CREATE TABLE table-name1
 (TYPE        CHAR(1) NOT NULL,
  SOURCE_RID  CHAR(5) NOT NULL,
  TARGET_XRID CHAR(9) NOT NULL,
  LRSN        CHAR(6) NOT NULL);
CREATE UNIQUE INDEX index-name1 ON table-name1
 (SOURCE_RID ASC, TYPE, TARGET_XRID, LRSN);
```

The REORG utility removes all rows from the mapping table when the utility completes.

You must specify the TARGET_XRID column as CHAR(9), even though the RIDs are 5 bytes long.

You must have DELETE, INSERT, and UPDATE authorization on the mapping table.

You can run more than one REORG SHRLEVEL CHANGE job concurrently, either on separate table spaces or on different partitions of the same table space. When you run concurrently with other jobs, each REORG job must have a separate mapping table. The mapping tables do not need to reside in separate table spaces. If only one mapping table exists, the REORG jobs must be scheduled to run serially. If more than one REORG job tries to access the same mapping table at the same time, one of the REORG jobs fails.

Recommendation: Consider the following approach to ensure that multiple REORG jobs do not attempt to use the same mapping table concurrently. Assign the same name to the mapping table and the utility ID. Because utility IDs must be unique, this naming decision ensures that the mapping tables are not used by two REORG jobs that run concurrently.

For a sample of using REORG with SHRLEVEL CHANGE and a sample mapping table and index, see job sample DSNTEJ1 in *DB2 Installation Guide*.

Restrictions when running REORG TABLESPACE on encrypted data

If you plan to run REORG TABLESPACE on encrypted data, do not use the WHEN statement to filter encrypted fields; REORG TABLESPACE cannot filter rows that contain encrypted data

Restrictions when using REBALANCE

Restriction for partitions with the COMPRESS YES attribute: Do not run REORG REBALANCE on a partitioned table space where a subset of partitions have the COMPRESS YES attribute and the remaining partitions have the COMPRESS NO attribute.

Restriction when duplicate partitioning key values exist: A REORG REBALANCE might distribute rows among the partitions that are being rebalanced in such a way that one or more partitions do not have any rows. This situation occurs when many rows with duplicate partitioning key values exist, and not enough unique values exist to enable REORG to distribute them over all of the partitions.

Restriction when physical partition numbers do not match logical partition numbers: A REORG REBALANCE might not be possible if the logical and physical partition numbers for the specified table space do not match. This situation can be created by a series of ALTER ROTATEs and ALTER ADD PARTs.

For example, assume that you create a table space with three partitions. Table 74 shows the mapping that exists between the physical and logical partition numbers.

Table 74. Mapping of physical and logical partition numbers when a table space with three partitions is created.

Logical partition number	Physical partition number
1	1
2	2
3	3

Then, assume that you request the following series of actions:

1. ALTER ROTATE FIRST TO LAST

   The new mapping of partition numbers is shown in Table 75

Table 75. Mapping of physical and logical partition numbers after ALTER ROTATE FIRST TO LAST.

Logical partition number	Physical partition number
1	2
2	3
3	1

2. ALTER ADD PART

   The new mapping of partition numbers is shown in Table 76.

Table 76. Mapping of physical and logical partition numbers after ALTER ADD PART.

Logical partition number	Physical partition number
1	2
2	3
3	1
4	4

3. ALTER ROTATE FIRST TO LAST

   The new mapping of partition numbers is shown in Table 77.

Table 77. Mapping of physical and logical partition numbers after second ALTER ROTATE FIRST TO LAST.

Logical partition number	Physical partition number
1	3

Table 77. Mapping of physical and logical partition numbers after second ALTER ROTATE FIRST TO LAST. (continued)

Logical partition number	Physical partition number
2	1
3	4
4	2

Assume that you then try to execute a REORG TABLESPACE REBALANCE PART 1:2. This statement requests a reorganization and rebalancing of physical partitions 1 and 2. Note that physical partition 1 is logical partition 2, and physical partition 2 is logical partition 4. Thus, the utility is processing logical partitions 2 and 4. If during the course of rebalancing, the utility needs to move keys from logical partition 2 to logical partition 3, the job fails, because logical partition 3 is not within the specified physical partition range.

Restart-pending status and SHRLEVEL CHANGE: If you specify SHRLEVEL CHANGE, REORG drains the write claim class near the end of REORG processing. In a data sharing environment, if a data sharing member fails and that member has restart-pending status for a target page set, the drain can fail. You must postpone running REORG with SHRLEVEL CHANGE until all restart-pending statuses are removed. You can use the DISPLAY GROUP command to determine whether a member's status is failed. You can use the DISPLAY DATABASE command with the LOCKS option to determine if locks are held.

RECOVER-pending and REBUILD-pending status: You cannot reorganize a table space if any partition or range of partitions of the partitioned table space is in the RECOVER-pending status. Similarly, you cannot reorganize a single table space partition if any of the following conditions are true:

- The partition is in the RECOVER-pending status.
- The corresponding partitioning index is in the REBUILD-pending or RECOVER-pending status, and the data is unloaded by the cluster index method.
- The specified partition or partitions are a subset of a range of partitions that are in REORG-pending status; you must reorganize the entire range to reset the restrictive status.

The only RECOVER-pending restrictive state is:

RECP The table space, index space, or partition of a table space or index space is in a RECOVER-pending status. A single logical partition in RECP does not restrict access to other logical partitions that are not in RECP. You can reset RECP by recovering only the single logical partition.

The three REBUILD-pending restrictive states are:

RBDP REBUILD-pending status is set on a physical or logical index partition. The individual physical or logical partition is inaccessible and must be rebuilt by using the REBUILD INDEX utility.

PSRBD
Page set REBUILD-pending status is set for nonpartitioning indexes. The entire index space is inaccessible and must be rebuilt by using the REBUILD utility.

RBDP*

A REBUILD-pending status that is set only on logical partitions of nonpartitioning indexes. The entire index is inaccessible, but it is made available again when the affected partitions are rebuilt by using the REBUILD INDEX utility.

For information about resetting the REBUILD-pending and RECOVER-pending states, see Table 170 on page 857 and Table 171 on page 857.

CHECK-pending status: If a table space is in both REORG-pending and CHECK-pending status (or auxiliary CHECK-pending status), run REORG first, and then run CHECK DATA to clear the respective states. Otherwise, if a table space is not in REORG-pending status, you cannot reorganize a table space or range of partitions if the table space or any partition in the range is in CHECK-pending status until the CHECK-pending status is removed. See "CHECK-pending status" on page 854 for more information about resetting the CHECK-pending status.

REORG-pending status: You must allocate a discard data set (SYSDISC) or specify the DISCARDDN option if the last partition of the table space is in REORG-pending status.

Data sets that REORG TABLESPACE uses

Table 78 describes the data sets that REORG TABLESPACE uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set, and any optional data sets that you want to use.

Table 78. Data sets that REORG TABLESPACE uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
STPRIN01	A data set that contains messages from DFSORT (usually, SYSOUT or DUMMY). This data set is used when frequency statistics are collected on DPSI's or when TABLESPACE TABLE COLGROUP FREQVAL is specified	No[1]
SYSDISC	Data set that contains discarded records from REORG DISCARD. The default DD name is SYSDISC.	No[2]
SYSPUNCH	Data set that contains a LOAD statement that is generated by REORG, which loads records that REORG DISCARD or REORG UNLOAD EXTERNAL wrote to the DISCARD or UNLOAD data sets. The default DD name is SYSPUNCH.	No[3]

Table 78. Data sets that REORG TABLESPACE uses (continued)

Data set	Description	Required?
Unload data set	Data set that contains the unloaded data that is to be reloaded during the RELOAD phase. Specify its DD or template name with the UNLDDN option or with the RECDSN field on the DB2I Utilities panel. The data set must be a sequential data set that is readable by BSAM. The default DD name is SYSREC. The unload data set must be large enough to contain all the unloaded records from all the tables in the target table space.If at least one table in the table space does not have an index, REORG cannot use the SORTDATA method with SHRLEVEL CHANGE. As a result, you must unload the data in the SYSREC data set.	Yes[4]
Copies	From one to four output data sets that are to contain the image copies. Specify their DD or template names with the COPYDDN and RECOVERYDDN options of the utility control statement.	No[5]
Work data sets	Temporary data sets for sort input and output. The DD names have the form DATAWK*nn*.	No[6]
Work data sets	Temporary data sets for sort input and output when sorting keys, or for sorting data when SORTDATA is specified but NOSYSREC is not. If index build parallelism is used, the DD names have the form SW*nn*WK*mm*. If index build parallelism is not used, the DD names have the form SORTWK*nn*	Yes[7]
Sort work data sets	Temporary data sets for sort input and output when collecting inline statistics on at least one data-partitioned secondary index. The DD names have the form ST01WK*nn*.	No[1, 8, 9]

Table 78. Data sets that REORG TABLESPACE uses (continued)

Data set	Description	Required?

Notes:

1. Required when collecting inline statistics on at least one data-partitioned secondary index.
2. Required if you specify DISCARDDN
3. Required you specify PUNCHDDN
4. Required unless NOSYSREC or SHRLEVEL CHANGE is specified.
5. Required if a partition is in REORG-pending status or REBALANCE, COPYDDN, RECOVERYDDN, SHRLEVEL REFERENCE, or SHRLEVEL CHANGE is specified.
6. Required if NOSYSREC or SHRLEVEL CHANGE is specified, but SORTDEVT is not specified.
7. Required if any indexes exist and SORTDEVT is not specified.
8. If the DYNALLOC parm of the SORT program is not turned on, you need to allocate the data set. Otherwise, DFSORT dynamically allocates the temporary data set.
9. It is recommended that you use dynamic allocation by specifying SORTDEVT in the utility statement because dynamic allocation reduces the maintenance required of the utility job JCL.

The following objects are named in the utility control statement and do not require DD statements in the JCL:

Table space
Object that is to be reorganized.

Calculating the size of the unload data set: The required size for the unload data set varies depending on the options that you use for REORG.

1. If you use REORG with UNLOAD PAUSE or CONTINUE and you specify KEEPDICTIONARY (assuming that a compression dictionary already exists), the size of the unload data set, in bytes, is the VSAM high-allocated RBA for the table space. You can obtain the high-allocated RBA from the associated VSAM catalog.

   For SHRLEVEL CHANGE, also add the result of the following calculation (in bytes) to the VSAM high-used RBA:

   ```
   number of records * 11
   ```

2. If you use REORG with UNLOAD ONLY, UNLOAD PAUSE, or CONTINUE and you do not specify KEEPDICTIONARY, you can calculate the size of the unload data set, in bytes, by using the following formula:

   ```
   maximum row length * number of rows
   ```

   The maximum row length is the row length, including the 6-byte record prefix, plus the length of the longest clustering key. If multiple tables exist in the table space, use the following formula to determine the maximum row length:

   ```
   Sum over all tables (row length * number of rows)
   ```

   For SHRLEVEL CHANGE, also add the result of the following formula to the preceding result:

   ```
   (21 * ((NEARINDREF + FARINDREF) * 1.1))
   ```

   In the preceding formula:

NEARINDREF
Is the value that is obtained from the NEARINDREF column of the SYSIBM.SYSTABLEPART catalog table.[3]

FARINDREF Is the value that is obtained from the FARINDREF column of the SYSIBM.SYSTABLEPART catalog table.

3. If you have variable-length fields, the calculation in step 2 on page 455 might result in excessive space. Use the average uncompressed row length, multiplied by the number of rows.
4. If you use REORG with UNLOAD PAUSE or CONTINUE with the DISCARD option, and the table has variable length fields, use the maximum row length in the calculation. The DISCARD option without the NOPAD option pads the variable length fields.

For certain table spaces in the catalog and directory, the unload data set for the table spaces have a different format. The calculation for the size of this data set is as follows:

```
data set size in bytes = (28 + longrow) * numrows
```

In the preceding formula:

longrow Is the length of the longest row in the table space.

numrows Is the number of rows in the data set.

The length of the row is calculated as follows:

```
Sum of column lengths + 4 bytes for each link
```

The length of the column is calculated as follows:

```
Maximum length of the column + 1 (if nullable) + 2 (if varying length)
```

See “Reorganizing the catalog and directory” on page 466 for more information about reorganizing catalog and directory table spaces.

Calculating the size of the work data sets: Allocating twice the space that is used by the input data sets is usually adequate for the sort work data sets. For compressed data, double again the amount of space that is allocated for the sort work data sets if you use either of the following REORG options:

- UNLOAD PAUSE without KEEPDICTIONARY
- UNLOAD CONTINUE without KEEPDICTIONARY

Using two or three large SORTWK*nn* data sets is preferable to using several small ones. If adequate space is not available, you cannot run REORG.

Specifying a destination for DFSORT messages: The REORG utility job step must contain a UTPRINT DD statement that defines a destination for messages that are issued by DFSORT during the SORT phase of REORG. DB2I, the %DSNU CLIST command, and the DSNUPROC procedure use the following default DD statement:

```
//UTPRINT DD  SYSOUT=A
```

Calculating the size of the sort work data sets: To calculate the approximate size (in bytes) of the ST01WK*nn* data set, use the following formula:

2 ×(*maximum record length* × *numcols* × (*count* + 2) × number of indexes)

3. The accuracy of the data set size calculation depends on recent information in the SYSTABLEPART catalog table.

The variables in the preceding formula have the following values:

maximum record length
Maximum record length of the SYSCOLDISTSTATS record that is processed when collecting frequency statistics (You can obtain this value from the RECLENGTH column in SYSTABLES.)

numcols
Number of key columns to concatenate when you collect frequent values from the specified index.

count Number of frequent values that DB2 is to collect.

DB2 utilities uses DFSORT to perform sorts. Sort work data sets cannot span volumes. Smaller volumes require more sort work data sets to sort the same amount of data; therefore, large volume sizes can reduce the number of needed sort work data sets. It is recommended that at least 1.2 times the amount of data to be sorted be provided in sort work data sets on disk. For more information about DFSORT, see *DFSORT Application Programming Guide*.

Shadow data sets

When you execute the REORG utility with SHRLEVEL REFERENCE or SHRLEVEL CHANGE, the utility uses shadow data sets.

For user-managed data sets, you must preallocate the shadow data sets before you execute REORG with SHRLEVEL REFERENCE or SHRLEVEL CHANGE. If a table space, partition, or index resides in DB2-managed data sets and shadow data sets do not already exist when you execute REORG, DB2 creates the shadow data sets. At the end of REORG processing, the DB2-managed shadow data sets are deleted.

Shadow data set names: Each shadow data set must have the following name:

catname.DSNDB*x*.*dbname*.*psname*.*y*0001.*Lnnn*

In the preceding name, the variables have the following meanings:

variable	meaning
catname	The VSAM catalog name or alias
x	C or D
dbname	Database name
psname	Table space name or index name
y	I or J
Lnnn	Partition identifier. Use one of the following values: • A001 through A999 for partitions 1 through 999 • B000 through B999 for partitions 1000 through 1999 • C000 through C999 for partitions 2000 through 2999 • D000 through D999 for partitions 3000 through 3999 • E000 through E996 for partitions 4000 through 4096

To determine the names of existing shadow data sets, execute one of the following queries against the SYSTABLEPART or SYSINDEXPART catalog tables:

```
SELECT DBNAME, TSNAME, IPREFIX
  FROM SYSIBM.SYSTABLEPART
  WHERE DBNAME = 'dbname' AND TSNAME = 'psname';
SELECT DBNAME, IXNAME, IPREFIX
  FROM SYSIBM.SYSINDEXES X, SYSIBM.SYSINDEXPART Y
  WHERE X.NAME = Y.IXNAME AND X.CREATOR = Y.IXCREATOR
  AND X.DBNAME = 'dbname' AND X.INDEXSPACE = 'psname';
```

For a partitioned table space, DB2 returns rows from which you select the row for the partitions that you want to reorganize.

For example, assume that you have a ten-partition table space and you want to determine a naming convention for the data set in order to successfully execute the REORG utility with the SHRLEVEL CHANGE PART 2:6 options. The following queries of the DB2 catalog tables SYSTABLEPART and SYSINDEXPART provide the required information:

```
SELECT DBNAME, TSNAME, PARTITION, IPREFIX FROM SYSIBM.SYSTABLEPART
  WHERE DBNAME = 'DBDV0701' AND TSNAME = 'TPDV0701'
  ORDER BY PARTITION;
SELECT IXNAME, PARTITION, IPREFIX FROM SYSIBM.SYSINDEXPART
  WHERE IXNAME = 'IXDV0701
  ORDER BY PARTITION;
```

The preceding queries produce the information that is shown in Table 79 and Table 80.

Table 79 shows the results from the first query.

Table 79. Query results from the first preceding query

DBNAME	TSNAME	PARTITION	IPREFIX
DBDV0701	TPDV0701	1	I
DBDV0701	TPDV0701	4	I
DBDV0701	TPDV0701	3	J
DBDV0701	TPDV0701	2	I
DBDV0701	TPDV0701	5	J
DBDV0701	TPDV0701	6	J
DBDV0701	TPDV0701	7	I
DBDV0701	TPDV0701	8	I
DBDV0701	TPDV0701	9	I
DBDV0701	TPDV0701	10	I

Table 80 shows the results from the second query.

Table 80. Query results from the second preceding query

IXNAME	PARTITION	IPREFIX
IXDV0701	10	I
IXDV0701	9	I
IXDV0701	8	I
IXDV0701	7	I
IXDV0701	6	J
IXDBV0701	5	J

Table 80. Query results from the second preceding query (continued)

IXNAME	PARTITION	IPREFIX
IXDV0701	4	I
IXDV0701	3	J
IXDV0701	2	I
IXDV0701	1	I

To execute REORG SHRLEVEL CHANGE PART 2:6, you need to preallocate the following shadow objects. The naming convention for these objects use information from the query results that are shown in Table 79 on page 458 and Table 80 on page 458.

```
vcatnam.DSNDBC.DBDV0701.TPDV0701.J0001.A002
vcatnam.DSNDBC.DBDV0701.TPDV0701.I0001.A003
vcatnam.DSNDBC.DBDV0701.TPDV0701.J0001.A004
vcatnam.DSNDBC.DBDV0701.TPDV0701.I0001.A005
vcatnam.DSNDBC.DBDV0701.TPDV0701.I0001.A006
vcatnam.DSNDBC.DBDV0701.IXDV0701.J0001.A002
vcatnam.DSNDBC.DBDV0701.IXDV0701.I0001.A003
vcatnam.DSNDBC.DBDV0701.IXDV0701.J0001.A004
vcatnam.DSNDBC.DBDV0701.IXDV0701.I0001.A005
vcatnam.DSNDBC.DBDV0701.IXDV0701.I0001.A006
```

Defining shadow data sets: Consider the following actions when you preallocate the data sets:

- Allocate the shadow data sets according to the rules for user-managed data sets.
- Define the shadow data sets as LINEAR.
- Use SHAREOPTIONS(3,3).
- Define the shadow data sets as EA-enabled if the original table space or index space is EA-enabled.
- Allocate the shadow data sets on the volumes that are defined in the storage group for the original table space or index space.

If you specify a secondary space quantity, DB2 does not use it. Instead, DB2 uses the SECQTY value for the table space or index space.

Recommendation: Use the MODEL option, which causes the new shadow data set to be created like the original data set. This method is shown in the following example:

```
DEFINE CLUSTER +
   (NAME('catname.DSNDBC.dbname.psname.x0001.L001') +
   MODEL('catname.DSNDBC.dbname.psname.y0001.L001')) +
   DATA                                              +
   (NAME('catname.DSNDBD.dbname.psname.x0001.L001') +
   MODEL('catname.DSNDBD.dbname.psname.y0001.L001') )
```

DB2 treats preallocated shadow data sets as DB2-managed data sets. For example, DB2 deletes a preallocated shadow data set for a nonpartitioning index at the end of REORG PART.

Creating shadow data sets for indexes: When you preallocate data sets for indexes, create the shadow data sets as follows:

- Create shadow data sets for the partition of the table space and the corresponding partition in each partitioning index and data-partitioned secondary index.

- Create a shadow data set for logical partitions of nonpartitioned secondary indexes.

Use the same naming scheme for these index data sets as you use for other data sets that are associated with the base index, except use J0001 instead of I0001. For more information about this naming scheme, see the information about the shadow data set naming convention at the beginning of this section, "Shadow data sets" on page 457.

Estimating the size of shadow data sets: If you have not changed the value of FREEPAGE or PCTFREE, the amount of required space for a shadow data set is comparable to the amount of required space for the original data set. However, for REORG PART, the required space for the shadow data set of the logical partition of a nonpartitioning index is approximately equal to the percentage of space that the partition occupies in the entire table space.

For example, a partitioned table space with 100 partitions and data that is relatively evenly balanced across the partitions needs a shadow data set for the logical partition that is approximately 1% of the size of the original nonpartitioning index.

Preallocating shadow data sets for REORG PART: By creating the shadow data sets before executing REORG PART, even for DB2-managed data sets, you prevent possible over-allocation of the disk space during REORG processing. When reorganizing a partition, you must create the shadow data sets for the partition of the table space and for the partition of the partitioning index. In addition, before executing REORG PART with SHRLEVEL REFERENCE or SHRLEVEL CHANGE on partition *mmm* of a partitioned table space, you must create a shadow data set for each nonpartitioning index that resides in user-defined data sets. Each shadow data set is to be used for a copy of the logical partition of the index. The name for this shadow data set has the form *catname*.DSNDB*x*.*dbname*.*psname*.y0*mmm*.A*nnn*.

When reorganizing a range of partitions, you must allocate a single shadow data set for each logical partition. Each logical partition within the range specified is contained in the single shadow data set. The name for this shadow data set must have the form *catname*.DSNDB*x*.*dbname*.*psname*.y0*mmm*.A*nnn*, where mmm is the first partition in the range specification.

Creating the control statement

Create the utility control statement for the REORG TABLESPACE job. See "Syntax and options of the REORG TABLESPACE control statement" on page 420 for REORG TABLESPACE syntax and option descriptions. See "Sample REORG TABLESPACE control statements" on page 486 for examples of REORG TABLESPACE usage.

Beginning in Version 8, the SORTKEYS option is the default. Therefore, the REORG TABLESPACE utility does not require SYSUT1 and SORTOUT data sets. The WORKDDN keyword, which provided the DD names of the SYSUT1 and SORTOUT data sets in earlier versions of DB2, is not needed and is ignored. The SORTKEYS keyword is also ignored. You do not need to modify existing control statements to remove the WORKDDN keyword or the SORTKEYS keyword.

Recommendation: Remove the DD statements from the job to prevent allocation of space that is not used.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in your utility control statement:

Determining when an object should be reorganized

Product-sensitive Programming Interface

You can determine when to run REORG for non-LOB table spaces and indexes by using the OFFPOSLIMIT and INDREFLIMIT catalog query options. If you specify the REPORTONLY option, REORG produces a report that indicates whether a REORG is recommended; a REORG is not performed.

When you specify the catalog query options along with the REPORTONLY option, REORG produces a report with one of the following return codes:

1 No limit met; no REORG is performed or recommended.
2 REORG is performed or recommended.

Alternatively, use the SYSTABLEPART and SYSINDEXPART catalog tables to find which table spaces and indexes qualify for reorganization. The information in these catalog tables can also be used to determine when the DB2 catalog table spaces require reorganization. For catalog table spaces SYSDBASE, SYSVIEWS, and SYSPLAN, you should not use the value for columns FAROFFPOSF and NEAROFFPOSF of SYSINDEXPART to determine whether to reorganize.

Table spaces or partitions that are in REORG-pending status should be reorganized. Use the DISPLAY DATABASE RESTRICT command to display those table spaces and partitions that require reorganization. See Appendix C, "Advisory or restrictive states," on page 853 for more information.

Information from the SYSTABLEPART catalog table can also tell you how well disk space is being used. If you want to find the number of varying-length rows that were relocated to other pages because of an update, run RUNSTATS, and then issue the following statement:

```
SELECT CARD, NEARINDREF, FARINDREF
  FROM SYSIBM.SYSTABLEPART
  WHERE DBNAME = 'XXX'
  AND TSNAME = 'YYY';
```

A large number (relative to previous values that you have received) for FARINDREF indicates that I/O activity on the table space is high. If you find that

this number increases over a period of time, you probably need to reorganize the table space to improve performance, and increase PCTFREE or FREEPAGE for the table space with the ALTER TABLESPACE statement.

The following statement returns the percentage of unused space in nonsegmented table space YYY. In nonsegmented table spaces, the space that is used by dropped tables is not reclaimed until you reorganize the table space.

```
SELECT PERCDROP
  FROM SYSIBM.SYSTABLEPART
  WHERE DBNAME = 'XXX'
  AND TSNAME = 'YYY';
```

Issue the following statement to determine whether the rows of a table are stored in the same order as the entries of its clustering index:

```
SELECT NEAROFFPOSF, FAROFFPOSF
  FROM SYSIBM.SYSINDEXPART
  WHERE IXCREATOR = 'index_creator_name'
  AND IXNAME = 'index_name';
```

Several indicators are available to signal a time for reorganizing table spaces. A large value for FAROFFPOSF might indicate that clustering is deteriorating. In this case, reorganizing the table space can improve query performance.

A large value for NEAROFFPOSF might indicate also that reorganization might improve performance. However, in general NEAROFFPOSF is not as critical a factor as FAROFFPOSF.

FAROFFPOSF and NEAROFFPOSF do not have query performance considerations for the following DB2 catalog tables:

DSNDB06.SYSDBASE
DSNDB06.SYSDBAUT
DSNDB06.SYSGROUP
DSNDB06.SYSPLAN
DSNDB06.SYSVIEWS

For any table, the REORG utility repositions rows into the sequence of the key of the clustering index that is defined on that table.

For nonclustering indexes, the statistical information that is recorded by RUNSTATS in SYSINDEXES and SYSINDEXPART might appear even worse after the clustering index is used to reorganize the data. This applies only to the CLUSTERING and CLUSTERED columns in SYSINDEXES and to the NEAROFFPOS and FAROFFPOS columns in SYSINDEXPART.

For specific REORG threshold numbers, see Part 5 of *DB2 Administration Guide*.

Recommendation: Run RUNSTATS if the statistics are not current. If you have an object that should also be reorganized, run REORG with STATISTICS and take inline copies. If you run REORG PART and nonpartitioning indexes exist, subsequently run RUNSTATS for each nonpartitioning index.

End of Product-sensitive Programming Interface

Specifying access with SHRLEVEL

For reorganizing a table space, or a partition of a table space, the SHRLEVEL option lets you choose the level of access that you have to your data during reorganization.

REORG with SHRLEVEL NONE, the default, reloads the reorganized data into the original area that is being reorganized. Applications have read-only access during unloading and no access during reloading. For data-partitioned secondary indexes, the option rebuilds the index parts during the BUILD phase. (Rebuilding these indexes does not create contention between parallel REORG PART jobs.) For nonpartitioned secondary indexes, the option corrects the indexes. Using REORG SHRLEVEL NONE is the only access level that resets REORG-pending status.

REORG with SHRLEVEL REFERENCE reloads the reorganized data into a new (shadow) copy of the area that is being reorganized. Near the end of reorganization, DB2 switches the future access of the application from the original data to the shadow copy. For SHRLEVEL REFERENCE, applications have read-only access during unloading and reloading, and a brief period of no access during switching. For data-partitioned secondary indexes, nothing occurs during the BUILD phase. (Rebuilding these indexes does not create contention between parallel REORG PART jobs.) For nonpartitioned secondary indexes, the option corrects the indexes of the reorganized parts.

REORG with SHRLEVEL CHANGE reloads the reorganized data into a shadow copy of the area that is being reorganized. For REORG TABLESPACE SHRLEVEL CHANGE, a mapping table correlates RIDs in the original copy of the table space or partition with RIDs in the shadow copy; see "Mapping table with SHRLEVEL CHANGE" on page 450 for instructions on creating the mapping table.

Applications can read from and write to the original area, and DB2 records the writing in the log. DB2 then reads the log and applies it to the shadow copy to bring the shadow copy up to date. This step executes iteratively, with each iteration processing a sequence of log records.

Near the end of reorganization, DB2 switches the future access of the application from the original data to the shadow copy. Applications have read-write access during unloading and reloading, a brief period of read-only access during the last iteration of log processing, and a brief period of no access during switching.

For data-partitioned secondary indexes, nothing occurs during the BUILD phase. (Rebuilding these indexes does not create contention between parallel REORG PART jobs.) For nonpartitioned secondary indexes, the option corrects the indexes of the reorganized parts.

Log processing with SHRLEVEL CHANGE: When you specify SHRLEVEL CHANGE, DB2 processes the log to update the shadow copy. This step executes iteratively. The first iteration processes the log records that accumulated during the previous iteration. The iterations continue until one of these conditions is met:

- DB2 estimates that the time to perform the log processing in the next iteration will be less than or equal to the time that is specified for MAXRO. If this condition is met, the next iteration is the last iteration.
- DB2 estimates that the SWITCH phase will not start by the deadline that is specified for DEADLINE. If this condition is met, DB2 terminates reorganization.
- The number of log records that the next iteration is to process is not sufficiently lower than the number of log records that were processed in the previous iteration. If this condition is met but the first two conditions are not met, DB2 sends message DSNU377I to the console. DB2 continues log processing for the length of time that is specified for DELAY and then performs the action that is specified for LONGLOG.

Operator actions: LONGLOG specifies the action that DB2 performs if the pace of processing log records between iterations is slow. See "Option descriptions" on page 426 for a description of the LONGLOG options. If no action is taken after message DSNU377I is sent to the console, the LONGLOG option automatically goes into effect. Some examples of possible actions that you can take:

- Execute the START DATABASE(*database*) SPACENAM(*tablespace*) ... ACCESS(RO) command and the QUIESCE utility to drain the write claim class. DB2 performs the last iteration, if MAXRO is not DEFER. After the QUIESCE, you should also execute the ALTER UTILITY command, even if you do not change any REORG parameters.
- Execute the START DATABASE(*database*) SPACENAM(*tablespace*) ... ACCESS(RO) command and the QUIESCE utility to drain the write claim class. Then, after reorganization makes some progress, execute the START DATABASE(*database*) SPACENAM(*tablespace*) ... ACCESS(RW) command. This increases the likelihood that processing of log records between iterations can continue at an acceptable rate. After the QUIESCE, you should also execute the ALTER UTILITY command, even if you do not change any REORG parameters.
- Execute the ALTER UTILITY command to change the value of MAXRO. Changing it to a huge positive value, such as 9999999, causes the next iteration to be the last iteration.
- Execute the ALTER UTILITY command to change the value of LONGLOG.
- Execute the TERM UTILITY command to terminate reorganization.
- Adjust the amount of buffer space that is allocated to reorganization and to applications. This adjustment can increase the likelihood that processing of log records between iterations can continue at an acceptable rate. After adjusting the space, you should also execute the ALTER UTILITY command, even if you do not change any REORG parameters.
- Adjust the scheduling priorities of reorganization and applications. This adjustment can increase the likelihood that processing of log records between iterations can continue at an acceptable rate. After adjusting the priorities, you should also execute the ALTER UTILITY command, even if you do not change any REORG parameters.

DB2 does not take the action that is specified in the LONGLOG phrase if any one of these events occurs before the delay expires:

- An ALTER UTILITY command is issued.
- A TERM UTILITY command is issued.
- DB2 estimates that the time to perform the next iteration is less than or equal to the time that is specified in the MAXRO keyword.
- REORG terminates for any reason (including the deadline).

Omitting the output data set

For REORG TABLESPACE, you can use the NOSYSREC option to omit the unload data set. You can use this option only if you do not specify UNLOAD PAUSE or UNLOAD ONLY. This option provides a performance advantage. However, you should be aware of the following facts:

- For REORG TABLESPACE SHRLEVEL CHANGE, REORG omits the unload data set, even if you do not specify NOSYSREC.
- For REORG TABLESPACE SHRLEVEL REFERENCE, if you do not use the NOSYSREC option and an error occurs during reloading, you can restart at the RELOAD phase of REORG by using the contents of the unload data set. However, if the REORG job includes both SORTDATA and NOSYSREC, you must restart at the UNLOAD phase.

- For REORG TABLESPACE SHRLEVEL NONE with NOSYSREC, if an error occurs during reloading, you must execute the RECOVER TABLESPACE utility, starting from the most recent image copy. Therefore, if you specify NOSYSREC with SHRLEVEL NONE, you must create an image copy before starting REORG TABLESPACE.

Unloading without reloading

REORG can unload data without continuing and without creating a SYSIBM.SYSUTIL record after the job ends.

If you specify UNLOAD ONLY, REORG unloads data from the table space and then ends. You can reload the data at a later date with the LOAD utility, specifying FORMAT UNLOAD.

Between unloading and reloading, you can add a validation routine to a table. During reloading, all the rows are checked by the validation procedure.

Do not use REORG UNLOAD ONLY to propagate data. When you specify the UNLOAD ONLY option, REORG unloads only the data that physically resides in the base table space; LOB columns are not unloaded. For purposes of data propagation, you should use UNLOAD or REORG UNLOAD EXTERNAL instead.

Reclaiming space from dropped tables

Reorganization omits tables that were previously dropped, reclaiming the space that they acquired. See “Reclaiming space in the DBD” on page 304 for actions to take when you drop a table.

Considerations for fallback recovery

If RECOVER cannot use the latest image copy or copies as a starting point for the recovery, it attempts to use previous copies; if that attempt fails, RECOVER restores the data from the log.

However, if you use REORG SHRLEVEL NONE LOG NO, RECOVER cannot restore data from the log past the point at which the object was last reorganized successfully. Therefore, you must take an image copy after running REORG with LOG NO to establish a level of fallback recovery.

Recommendation: Immediately following an ALTER INDEX operation that modifies key values, create a new recovery point by taking one of the following actions:

- Run REORG and specify COPYDDN and SHRLEVEL NONE.
- Take a full image copy immediately after REORG completes.

If you performed a REORG to reset REORG-pending status (REORP), you should also take an inline image copy or run the COPY utility. Image copies that are taken prior to resetting the REORG-pending status cannot be used for recovery to the current RBA or LRSN.

Successful REORG LOG NO processing inserts a row into SYSIBM.SYSCOPY with ICTYPE=W for each index that was defined with COPY YES. REORG also places a reorganized index in informational COPY-pending (ICOPY) status. You should take a full image copy of the index after the REORG job completes to create a valid point of recovery.

Reorganizing the catalog and directory

You can run REORG TABLESPACE on the table spaces in the catalog database (DSNDB06) and on the SCT02, SPT01, DBD01, and SYSLGRNX table spaces in the directory database (DSNDB01).

Attention: You must take a full image copy before and after reorganizing any catalog or directory object. Otherwise, you cannot recover any catalog of directory objects without the full image copies. When you reorganize the DSNDB06.SYSCOPY table space with the LOG NO option and omit the COPYDDN option, DB2 places the table space in COPY-pending status. Take a full image copy of the table space to remove the COPY-pending status before continuing to reorganize the catalog or directory table spaces.

Running REORG LOG NO COPYDDN avoids the COPY-pending status, because an inline copy is taken during the REORG.

The FASTSWITCH YES option is ignored for catalog and directory objects.

Recommendation: Use SHRLEVEL REFERENCE when reorganizing the catalog.

When to run REORG on the catalog and directory: You do not need to run REORG TABLESPACE on the catalog and directory table spaces as often as you do on user table spaces. RUNSTATS collects statistics about user table spaces which you use to determine if a REORG is necessary. You can use the same statistics to determine if a REORG is needed for catalog table spaces. The only difference is the information in the columns NEAROFFPOSF and FAROFFPOSF in table SYSINDEXPART. The values in these columns can be double the recommended value for user table spaces before a reorganization is needed if the table space is DSNDB06.SYSDBASE, DSNDB06.SYSVIEWS, DSNDB06.SYSPLAN, DSNDB06.SYSGROUP, or DSNDB06.SYSDBAUT.

Reorganize the whole catalog before a catalog migration or once every couple of years. Reorganizing the catalog is useful for reducing the size of the catalog table space. To improve query performance, reorganize the indexes on the catalog tables.

When statistical information indicates that DSNDB06.SYSDBASE, DSNDB06.SYSPLAN, or DSNDB06.SYSPKAGE requires reorganization, you should also reorganize the corresponding directory table space. These catalog table spaces and their corresponding directory table spaces are listed in Table 81.

Table 81. Catalog table spaces and their corresponding directory table spaces

Catalog table space	Directory table space
DSNDB06.SYSDBASE	DSNDB01.DBD01
DSNDB06.SYSPLAN	DSNDB01.SCT02
DSNDB06.SYSPKAGE	DSNDB01.SPT01

Associated directory table spaces: When certain catalog table spaces are reorganized, you should also reorganize the associated directory table space. The associated directory table spaces are listed in Table 81.

Limitations for reorganizing the catalog and directory:

- You cannot reorganize DSNDB01.SYSUTILX.

- The UNLOAD ONLY or UNLOAD EXTERNAL and LOG YES options are not allowed for catalog and directory table spaces. However, LOG YES is required for the catalog LOB table spaces.
- The SORTDEVT and SORTNUM options are ignored for the following catalog and directory table spaces:
  - DSNDB06.SYSDBASE
  - DSNDB06.SYSDBAUT
  - DSNDB06.SYSGROUP
  - DSNDB06.SYSPLAN
  - DSNDB06.SYSVIEWS
  - DSNDB01.DBD01

  The COPYDDN and RECOVERYDDN options are valid for the preceding catalog and directory tables if SHRLEVEL REFERENCE is also specified.
- REORG TABLESPACE with SHRLEVEL CHANGE cannot operate on the following catalog and directory table spaces:
  - DSNDB06.SYSDBASE
  - DSNDB06.SYSDBAUT
  - DSNDB06.SYSGROUP
  - DSNDB06.SYSPLAN
  - DSNDB06.SYSVIEWS
  - DSNDB01.DBD01
- REORG TABLESPACE with STATISTICS cannot collect inline statistics on the following catalog and directory table spaces:
  - DSNDB06.SYSDBASE
  - DSNDB06.SYSDBAUT
  - DSNDB06.SYSGROUP
  - DSNDB06.SYSPLAN
  - DSNDB06.SYSVIEWS
  - DSBDB06.SYSSTATS
  - DSNDB06.SYSHIST
  - DSNDB01.DBD01

Phases for reorganizing the catalog and directory: REORG TABLESPACE processes certain catalog and directory table spaces differently from other table spaces; it does not execute the BUILD and SORT phases for the following table spaces:

- DSNDB06.SYSDBASE
- DSNDB06.SYSDBAUT
- DSNDB06.SYSGROUP
- DSNDB06.SYSPLAN
- DSNDB06.SYSVIEWS
- DSNDB01.DBD01

For these table spaces, REORG TABLESPACE reloads the indexes (in addition to the table space) during the RELOAD phase, rather than storing the index keys in a work data set for sorting.

For all other catalog and directory table spaces, DB2 uses index build parallelism.

Changing data set definitions

If the table space is defined by storage groups, DB2 allocates space, and you cannot alter data set definitions during the reorganization process. DB2 deletes and redefines the necessary data sets to reorganize the object.

For REORG with SHRLEVEL REFERENCE or CHANGE, you can use the ALTER STOGROUP command to change the characteristics of a DB2-managed data set. To

change the characteristics of a user-managed data set, specify the desired new characteristics when you create the shadow data set; see "Shadow data sets" on page 457 for more information about user-managed data sets. For example, placing the original and shadow data sets on different disk volumes might reduce contention and thus improve the performance of REORG and the performance of applications during REORG execution.

Temporarily interrupting REORG

You can temporarily pause REORG. If you specify UNLOAD PAUSE, REORG pauses after unloading the table space into the unload data set. You cannot use NOSYSREC and PAUSE. The job completes with return code 4. You can restart REORG by using the phase restart or current restart. Do not alter the REORG statement.

The REORG utility remains in stopped status until REORG is restarted or terminated.

While REORG is interrupted by PAUSE, you can redefine the table space attributes for user-defined table spaces. PAUSE is not required for STOGROUP-defined table spaces. Attribute changes are done automatically by a REORG following an ALTER TABLESPACE.

Building a compression dictionary

The REORG utility builds the compression dictionary during the UNLOAD process. This dictionary is then used during the RELOAD phase to compress the data. Specify the KEEPDICTIONARY option to save the cost of rebuilding the dictionary if you are satisfied with the current compression ratio.

Overriding dynamic DFSORT and SORTDATA allocation

If your REORG job includes the SORTDATA option, DB2 estimates how many rows are to be sorted and passes this information to DFSORT on the parameter FILSZ. DFSORT then dynamically allocates the necessary sort work space.

If the table space contains rows with VARCHAR columns, DB2 might not be able to accurately estimate the number of rows. If the estimated number of rows is too high and the sort work space is not available or if the estimated number of rows is too low, DFSORT might fail and cause an abend. **Important:** Run RUNSTATS UPDATE SPACE before the REORG so that DB2 calculates a more accurate estimate.

You can override this dynamic allocation of sort work space in two ways:

- Allocate the sort work data sets with SORTWK*nn* DD statements in your JCL.
- Override the DB2 row estimate in FILSZ using control statements that are passed to DFSORT. However, using control statements overrides size estimates that are passed to DFSORT in all invocations of DFSORT in the job step, including sorting keys to build indexes, and any sorts that are done in any other utility that is executed in the same step. The result might be reduced sort efficiency or an abend due to an out-of-space condition.

Rebalancing partitions by using REORG

You can use the following methods to rebalance partitions:

- Use ALTER INDEX to modify the limit keys for partition boundaries before you use REORG TABLESPACE.
- Use ALTER TABLE ALTER PARTITION before you use REORG TABLESPACE.

- Use REBALANCE on the REORG TABLESPACE utility. **Restriction:** You cannot use the REBALANCE option with the SCOPE PENDING option.

If you use ALTER INDEX to modify the limit keys for partition boundaries, you must subsequently use REORG TABLESPACE to redistribute data in the partitioned table spaces based on the new key values and to reset the REORG-pending status. The following example specifies options that help maximize performance while performing the necessary rebalancing reorganization:

```
REORG TABLESPACE DSN8S81E PART 2:3
 NOSYSREC
 COPYDDN SYSCOPY
 STATISTICS TABLE INDEX(ALL)
```

You can reorganize a range of partitions, even if the partitions are not in REORG-pending status. If you specify the STATISTICS keyword, REORG collects data about the specified range of partitions.

If you perform a REORG on partitions that are in the REORG-pending status, be aware that:

- You must specify SHRLEVEL NONE if the object is in REORG-pending status. Otherwise, REORG terminates and issues message DSNU273I and return code 8.
- REORG ignores the KEEPDICTIONARY option for any partition that is in REORG-pending status; REORG automatically rebuilds the dictionaries for the affected partitions. However, if you specify a range of partitions that includes some partitions that are not in REORG-pending restrictive status, REORG honors the KEEPDICTIONARY option for those nonrestricted partitions.
- If any partition is in REORG-pending status when REORG executes, DB2 writes a SYSCOPY record with STYPE=A for each partition that is specified on the REORG job.
- If you take an inline image copy of a range of partitions, DB2 writes one SYSCOPY record with ICTYPE=F for each partition, and each record has the same data set name.
- Specify the DISCARDDN and PUNCHDDN data sets for a table space that is defined as LARGE or DSSIZE, but has had the limit key for the last partition of the table space reduced by a subsequent ALTER INDEX statement. Otherwise, REORG terminates and issues message DSNU035I and return code 8.

You cannot reorganize a subset of a range of partitions that are in REORG-pending status; you must reorganize the entire range to reset the restrictive status.

For more restrictions when using REBALANCE, see "Restrictions when using REBALANCE" on page 450.

Rebalancing partitions when the clustering index does not match the partitioning key: For a table that has a clustering index that does not match the partitioning key, you must run REORG TABLESPACE twice so that data is rebalanced and all rows are in clustering order. The first utility execution rebalances the data and the second utility execution sorts the data.

For example, assume you have a table space that was created with the following SQL:

```
-----------------------------------------
   SQL to create a table and index with
   separate columns for partitioning
   and clustering
-----------------------------------------
```

```
CREATE TABLESPACE TS IN DB
       USING STOGROUP SG
       NUMPARTS 4 BUFFERPOOL BP0;

CREATE TABLE TB (C01 CHAR(5) NOT NULL,
                 C02 CHAR(5) NOT NULL,
                 C03 CHAR(5) NOT NULL)
       IN DB.TS
          PARTITION BY (C01)
             (PART 1 VALUES ('00001'),
              PART 2 VALUES ('00002'),
              PART 3 VALUES ('00003'),
              PART 4 VALUES ('00004'));

CREATE INDEX IX ON TB(C02) CLUSTER;
```

To rebalance the data across the four partitions, use the following REORG TABLESPACE control statement:

```
REORG TABLESPACE DB.TS REBALANCE
```

After the preceding utility job completes, the table space is placed in AREO* status to indicate that a subsequent reorganization is recommended to ensure that the rows are in clustering order. For this subsequent reorganization, use the following REORG TABLESPACE control statement:

```
REORG TABLESPACE DB.TS
```

Using inline copy with REORG TABLESPACE

You can create a full image copy data set (SHRLEVEL REFERENCE) during REORG TABLESPACE execution. The new copy is an inline copy. The advantage to using an inline copy is that the table space is not left in COPY-pending status, regardless of which LOG option is specified for the utility. Thus, data availability is increased. You must take an inline copy when you specify the REBALANCE option. You cannot take inline copies of LOB table spaces.

To create an inline copy, use the COPYDDN and RECOVERYDDN keywords. You can specify up to two primary copies and two secondary copies. Inline copies are produced during the RELOAD phase of REORG processing.

The SYSCOPY record that is produced by an inline copy contains ICTYPE=F, SHRLEVEL=R. The STYPE column contains an X if the image copy was produced by REORG TABLESPACE LOG(YES), and a W if the image copy was produced by REORG TABLESPACE LOG(NO). The data set that is produced by the inline copy is logically equivalent to a full image copy with SHRLEVEL REFERENCE, but the data within the data set differs in some respects:

- Data pages might be out of sequence and some might be repeated. If pages are repeated, the last one is always the correct copy.
- Space map pages are out of sequence and might be repeated

The total number of duplicate pages is small, with a negligible effect on the amount of space that is required for the data set. One exception to this guideline is the case of running REORG SHRLEVEL CHANGE, in which the number of duplicate pages varies with the number of records that are applied during the LOG phase.

Note: If the inline copy dataset is allocated to TAPE, you need to hold the drain in switch phase until the inline copy dataset is closed and deallocated from the TAPE device.

Improving performance

To improve REORG performance:

- Run REORG concurrently on separate partitions of a partitioned table space. When you run REORG on partitions of a partitioned table space, the sum of each job's processor usage is greater than for a single REORG job on the entire table space. However, the elapsed time of reorganizing the entire table in parallel can be significantly less than it would be for a single REORG job.
- Use parallel index build for table spaces or partitions that have more than one defined index. For more information, see "Building indexes in parallel for REORG TABLESPACE" on page 472.
- Specify NOSYSREC on your REORG statement. See "Omitting the output data set" on page 464 for restrictions.
- If you are using 3990 caching, and you have the nonpartitioning indexes on RAMAC®, consider specifying YES on the UTILITY CACHE OPTION field of installation panel DSNTIPE. This option allows DB2 to use sequential prestaging when reading data from RAMAC for the following utilities:
  - LOAD PART *integer* RESUME
  - REORG TABLESPACE PART

  For LOAD PART and REORG TABLESPACE PART utility jobs, prefetch reads remain in the cache longer, which can lead to possible improvements in the performance of subsequent writes.

  Use inline copy and inline statistics instead of running separate COPY and RUNSTATS utilities.

When to use SHRLEVEL CHANGE: Schedule REORG with SHRLEVEL CHANGE when the rate of writing is low and transactions are short. Avoid scheduling REORG with SHRLEVEL CHANGE when critical applications are executing.

Performance implications with SHRLEVEL CHANGE: Under certain circumstances, the log records that REORG SHRLEVEL CHANGE uses contain additional information, as if DATA CAPTURE CHANGES were used. Generation of the additional information can slow applications and increase consumption of log space. The additional information is generated for all the tables in the table space if at least one table satisfies all these conditions:

- The table has undergone ALTER TABLE ADD column.
- The table does not use DATA CAPTURE CHANGES.
- One of these conditions is true:
  - The area that is being reorganized uses data compression.
  - The area is a partitioned table space, and at least one partition uses data compression.

When to use DRAIN_WAIT: The DRAIN_WAIT option gives you greater control over the time that online REORG is to wait for drains. Also because the DRAIN_WAIT is the aggregate time that online REORG is to wait to perform a drain on a table space and associated indexes, the length of drains is more predictable than if each partition and index has its own individual waiting time limit.

By specifying a short delay time (less than the system timeout value, IRLMRWT), you can reduce the impact on applications by reducing time-outs. You can use the RETRY option to give the online REORG more chances to complete successfully. If you do not want to use RETRY processing, you can still use DRAIN_WAIT to set a specific and more consistent limit on the length of drains.

RETRY allows an online REORG that is unable to drain the objects that it requires so that DB2 can try again after a set period (RETRY_DELAY). During the RETRY_DELAY period, all the objects are available for read-write access in the case of SHRLEVEL CHANGE. For SHRLEVEL REFERENCE, the objects remain with the access that existed prior to the attempted drain (that is if the drain fails in the UNLOAD phase the object remains in read-write access; if the drain fails in the SWITCH phase, objects remain in read-only access). Because application SQL statements can be in a queue behind any unsuccessful drain the online REORG has tried, a reasonable delay is recommended before retrying, to allow this work to complete; the default is 5 minutes.

When you specify DRAIN WRITERS (the default) with SHRLEVEL CHANGE and RETRY, multiple read-only log iterations can occur. Generally, online REORG might need to do more work when RETRY is specified, and this might result in multiple or extended periods of restricted access. Applications that run alongside online REORG need to perform frequent commits. During the interval between retries, the utility is still active, and consequently other utility activity against the table space and indexes is restricted.

When doing a table space REORG with RETRY and SHRLEVEL CHANGE both specified, you can increase the size of the COPY that REORG takes.

Recommendation: Run online REORG during lighter periods of activity on the table space or index.

Building indexes in parallel for REORG TABLESPACE

Parallel index build reduces the elapsed time for a REORG TABLESPACE job by sorting the index keys and rebuilding multiple indexes in parallel, rather than sequentially. Optimally, a pair of subtasks processes each index; one subtlest sorts extracted keys, while the other subtlest builds the index. REORG TABLESPACE begins building each index as soon as the corresponding sort emits its first sorted record.

Figure 78 on page 473 shows the flow of a REORG TABLESPACE job that uses a parallel index build. DB2 starts multiple subtasks to sort index keys and build indexes in parallel. If you specify STATISTICS, additional subtasks collect the sorted keys and update the catalog table in parallel, eliminating the need for a second scan of the index by a separate RUNSTATS job.

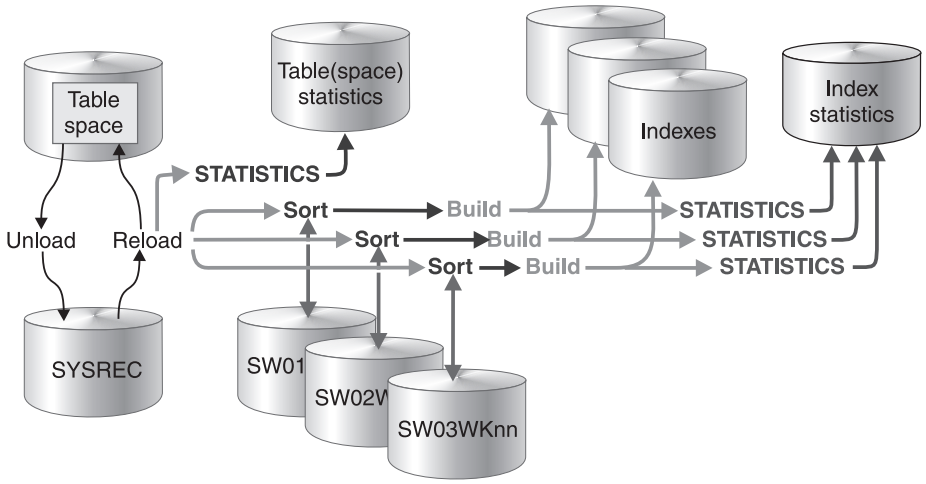


Figure 78. How indexes are built during a parallel index build

REORG TABLESPACE uses parallel index build if more than one index needs to be built (including the mapping index for SHRLEVEL CHANGE). You can either let the utility dynamically allocate the data sets that SORT needs for this parallel index build or provide the necessary data sets yourself.

Select one of the following methods to allocate sort work and message data sets:

Method 1: REORG TABLESPACE determines the optimal number of sort work data sets and message data sets.

1. Specify the SORTDEVT keyword in the utility statement.
2. Allow dynamic allocation of sort work data sets by **not** supplying SORTWK*nn* DD statements in the REORG TABLESPACE utility JCL.
3. Allocate UTPRINT to SYSOUT.

Method 2: Control allocation of sort work data sets, while REORG TABLESPACE allocates message data sets.

1. Provide DD statements with DD names in the form SW*nn*WK*mm*.
2. Allocate UTPRINT to SYSOUT.

Method 3: Exercise the most control over rebuild processing; specify both sort work data sets and message data sets.

1. Provide DD statements with DD names in the form SW*nn*WK*mm*.
2. Provide DD statements with DD names in the form UTPRIN*nn*.

Data sets used: If you select Method 2 or 3 in the preceding information, define the necessary data sets by using the information provided here, along with "Determining the number of sort subtasks" on page 474, "Allocation of sort subtasks" on page 474, and "Estimating the sort work file size" on page 474.

Each sort subtask must have its own group of sort work data sets and its own print message data set. Possible reasons to allocate data sets in the utility job JCL rather than using dynamic allocation are:

- To control the size and placement of the data sets
- To minimize device contention
- To optimally utilize free disk space
- To limit the number of utility subtasks that are used to build indexes

The DD name SW*nn*WK*mm* defines the sort work data sets that are used during utility processing. *nn* identifies the subtask pair, and *mm* identifies one or more data sets that are to be used by that subtask pair. For example:

SW01WK01 Is the first sort work data set that is used by the subtask that builds the first index.

SW01WK02 Is the second sort work data set that is used by the subtask that builds the first index.

SW02WK01 Is the first sort work data set that is used by the subtask that builds the second index.

SW02WK02 Is the second sort work data set that is used by the subtask that builds the second index.

The DD name UTPRIN*nn* defines the sort work message data sets that are used by the utility subtask pairs. *nn* identifies the subtask pair.

Determining the number of sort subtasks: The maximum number of utility subtask pairs that are started for parallel index build is equal to the number of indexes that need to be built.

REORG TABLESPACE determines the number of subtask pairs according to the following guidelines:

- The number of subtask pairs equals the number of allocated sort work data set groups.
- The number of subtask pairs equals the number of allocated message data sets.
- If you allocate both sort work data sets and message data set groups, the number of subtask pairs equals the smallest number of allocated data sets.

Allocation of sort subtasks: REORG TABLESPACE attempts to assign one sort subtask pair for each index that is to be built. If REORG TABLESPACE cannot start enough subtasks to build one index per subtask pair, it allocates any excess indexes across the pairs; therefore one or more subtask pairs might build more than one index.

During parallel index build processing, REORG distributes all indexes among the subtask pairs according to the index creation date, assigning the first created index to the first subtask pair. For SHRLEVEL CHANGE, the mapping index is assigned last.

Estimating the sort work file size: If you choose to provide the data sets, you need to know the size and number of keys that are present in all of the indexes that are being processed by the subtask in order to calculate each sort work file size. After you determine which indexes are assigned to which subtask pairs, use the following formula to calculate the required space:

2 × (longest index key + c) × (number of extracted keys)

longest key The length of the longest index key that is to be processed by the subtask. If the index is of varying length, the longest key is the maximum possible length of a key with all varying-length columns that are padded to their maximum length, plus 2 bytes for each varying-length column in the index. For example, if an index with three columns (A, B, and C) has length values of CHAR(8) for A, VARCHAR(128) for B, and VARCHAR(50) for C, the longest key is calculated as follows:

```
8 + 128 + 50 + 2 + 2 = 190
```

For SHRLEVEL CHANGE, the mapping index key length is 21.

c A value as follows:

- 10 if the indexes that are rebuilt are a mix of data-partitioned secondary indexes and nonpartitioned indexes
- 8 if all indexes are partitioned or none of them are data-partitioned secondary indexes.

number of keys The number of keys from all indexes that need to be sorted and that are to be processed by the subtask.

Do not count keys that belong to partitioning indexes should not be counted in the sort work data set size calculation. The space estimation formula might indicate that 0 bytes are required (because the only index that is processed by a task set is the partitioning index). In this case, if you allocate your own sort work data set groups, you still need to allocate sort work data sets for this task set, but you can use a minimal allocation, such as 1 track.

For information about LOAD and REORG performance, see "Improving performance with LOAD or REORG PREFORMAT" on page 254.

Methods of unloading data

DB2 unloads data by one of three methods:

- *Table space scan with sort:* If at least one table space has an index, DB2 uses a table-space scan with a sort.
- *Table space scan:* DB2 uses a table-space scan for simple table spaces that contain more than one table, or that contain one table but do not have an index.
- *Clustering index:* DB2 uses this option for simple table spaces that contain one table and have an index, and for tables in a segmented table space that have an index.

Encountering an error in the RELOAD phase

Failure during the RELOAD phase (after the data is unloaded and data sets are deleted, but before the data is reloaded) results in an unusable table space.

If the error is on the table space data:

- If you have defined data sets, you can allocate new data sets.
- If STOGROUP has defined data sets, you can alter the new table space to change the primary and secondary quantities.
- If you allocate new data sets, alter the table space, or add volumes to the storage group, restart the REORG job at the beginning of the phase. Otherwise, you can restart either at the last commit point or at the beginning of the phase.

If the error is on the unloaded data, or if you used the NOSYSREC option, terminate REORG by using the TERM UTILITY command. Then recover the table space, using RECOVER, and run the REORG job again.

Reorganizing partitioned table spaces

If you reorganize a single partition or a range of partitions, all indexes of the table space are affected. Depending on how disorganized the nonpartitioning indexes are, you might want to reorganize them, as well. For more information about when to reorganize, see "Determining when an index requires reorganization" on page 407.

Reorganizing segmented table spaces

If the target table space is segmented, REORG unloads and reloads by table.

If an index exists on a table in a segmented table space, that table is unloaded in clustering sequence. If NO index exists, the table is unloaded in physical row and segment order.

For segmented table spaces, REORG does **not** normally need to reclaim space from dropped tables. Space that is freed by dropping tables in a segmented table space is immediately available if the table space can be accessed when DROP TABLE is executed. If the table space cannot be accessed when DROP TABLE is executed (for example, the disk device is offline), DB2 removes the table from the catalog, but does not delete all table rows. In this case, the space for the dropped table is not available until REORG reclaims it.

After you run REORG, the segments for each table are contiguous.

Counting records loaded during RELOAD phase

At the end of the RELOAD phase, REORG compares the number of records that were actually loaded to the number of records that were unloaded. If the counts do not match, the resulting actions depend on the UNLOAD option that you specified on the original job:

- If you specify UNLOAD PAUSE, REORG sets return code 4 and continues processing the job.
- If you specify UNLOAD CONTINUE, DB2 issues an error message and abnormally terminates the job. The table space or partition remains in RECOVER-pending status.

Reorganizing a LOB table space

Reorganizing a LOB table space is a separate task from reorganizing the base table space. REORG does not unload LOBs, and it does not reclaim physical space. A LOB table space that is defined with LOG YES or LOG NO affects logging during the reorganizing a LOB column. Table 40 on page 263 shows the logging output and LOB table space effect, if any. SYSIBM.SYSCOPY is not updated.

Specify LOG YES and SHRLEVEL NONE when you reorganize a LOB table space to avoid leaving the LOB table space in COPY-pending status after the REORG.

Terminating or restarting REORG TABLESPACE

This section contains information about how to terminate and restart REORG TABLESPACE.

Terminating REORG TABLESPACE

If you terminate REORG TABLESPACE with the TERM UTILITY command during the UNLOAD phase, objects have not yet been changed, and you can rerun the job.

If you terminate REORG TABLESPACE with the TERM UTILITY command during the RELOAD phase, the behavior depends on the SHRLEVEL option:

- For SHRLEVEL NONE, the data records are not erased. The table space and indexes remain in RECOVER-pending status. After you recover the table space, rerun the REORG job.
- For SHRLEVEL REFERENCE or CHANGE, the data records are reloaded into shadow objects, so the original objects have not been affected by REORG. You can rerun the job.

If you terminate REORG with the TERM UTILITY command during the SORT, BUILD, or LOG phases, the behavior depends on the SHRLEVEL option:

- For SHRLEVEL NONE, the indexes that are not yet built remain in RECOVER-pending status. You can run REORG with the SORTDATA option, or you can run REBUILD INDEX to rebuild those indexes.
- For SHRLEVEL REFERENCE or CHANGE, the records are reloaded into shadow objects, so the original objects have not been affected by REORG. You can rerun the job.

If you terminate a stopped REORG utility with the TERM UTILITY command during the SWITCH phase, the following conditions apply:

- All data sets that were renamed to their shadow counterparts are renamed to their original names, so that the objects remain in their original state, and you can rerun the job.
- If a problem occurs in renaming the data sets to the original names, the objects remain in RECOVER-pending status, and you cannot rerun the job.

If the SWITCH phase does not complete, the image copy that REORG created is not available for use by the RECOVER utility. If you terminate an active REORG utility during the SWITCH phase with the TERM UTILITY command, during the rename process, the renaming occurs, and the SWITCH phase completes. The image copy that REORG created is available for use by the RECOVER utility.

If you terminate REORG with the TERM UTILITY command during the BUILD2 phase, the logical partitions of nonpartitioned indexes remain in RECOVER-pending status. After you run REBUILD INDEX for the logical partition, all objects have been reorganized successfully.

The REORG-pending status is not reset until the UTILTERM execution phase. If the REORG utility abnormally terminates or is terminated, the objects remain in REORG-pending status and RECOVER-pending status, depending on the phase in which the failure occurred. See Appendix C, “Advisory or restrictive states,” on page 853 for information about resetting either status.

Table 82 lists the restrictive states that REORG TABLESPACE sets according to the phase in which the utility terminated.

Table 82. Restrictive states that REORG TABLESPACE sets.

Phase	Effect on restrictive status
UNLOAD	No effect.
RELOAD	SHRLEVEL NONE: • Places table space in RECOVER-pending status at the beginning of the phase and resets the status at the end of the phase. • Places indexes in RECOVER-pending status. • Places the table space in COPY-pending status. If COPYDDN is specified and SORTKEYS is ignored, the COPY-pending status is reset at the end of the phase. SORTKEYS is ignored for several catalog and directory table spaces. For a list of these table spaces, see “Reorganizing the catalog and directory” on page 466. SHRLEVEL REFERENCE or CHANGE has no effect.
SORT	No effect.

Table 82. Restrictive states that REORG TABLESPACE sets. (continued)

Phase	Effect on restrictive status
BUILD	SHRLEVEL NONE resets RECOVER-pending status for indexes and, if the utility job includes both COPYDDN and SORTKEYS, resets COPY-pending status for table spaces at the end of the phase. SHRLEVEL REFERENCE or CHANGE has no effect.
SORTBLD	No effect during the sort portion of the SORTBLD phase. During the build portion of the SORTBLD phase, the effect is the same as for the BUILD phase.
LOG	No effect.
SWITCH	No effect. Under certain conditions, if TERM UTILITY is issued, it must complete successfully; otherwise, objects might be placed in RECOVER-pending status.
BUILD2	If TERM UTILITY is issued, the logical partitions for nonpartitioning indexes are placed in logical RECOVER-pending status.

Recovering a failed REORG job: If you terminate REORG SHRLEVEL NONE in the RELOAD phase, all SYSLGRNX records associated with the reorganization are deleted. Use the RECOVER TABLESPACE utility to recover to the current point in time. This action recovers the table space to its state before the failed reorganization.

Restarting REORG TABLESPACE

By default, DB2 uses RESTART(CURRENT) when restarting REORG TABLESPACE jobs, with the following exceptions:

- Jobs that are restarted in the SORT, BUILD, SWITCH, or BUILD2 phase use RESTART(PHASE) by default.
- Jobs with the SORTKEYS option that are restarted in the RELOAD, SORT, BUILD, or SORTBLD phase always restart from the beginning of the RELOAD phase.
- Jobs with the SHRLEVEL REFERENCE, NOSYSREC, and SORTDATA options use RESTART(PHASE) to restart at the beginning of the UNLOAD phase.
- Jobs that reorganize the following catalog or directory table spaces use RESTART(PHASE):
  - DSNDB06.SYSDBASE
  - DSNDB06.SYSDBAUT
  - DSNDB06.SYSGROUP
  - DSNDB06.SYSPLAN
  - DSNDB06.SYSVIEWS
  - DSNDB01.DBD01

If you restart a REORG job of one or more of the catalog or directory table spaces in the preceding list, you cannot use RESTART(CURRENT).

If you restart REORG in the UTILINIT phase, it re-executes from the beginning of the phase. If REORG abnormally terminates or system failure occurs while it is in the UTILTERM phase, you must restart the job with RESTART(PHASE).

Table 83 on page 479 provides information about restarting REORG TABLESPACE, depending on the phase that REORG was in when the job stopped.

For each phase of REORG and for each type of REORG TABLESPACE (with SHRLEVEL NONE, with SHRLEVEL REFERENCE, and with SHRLEVEL

CHANGE), Table 83 indicates the types of restarts that are allowed (CURRENT and PHASE). A value of None indicates that no restart is allowed. The ″Data Sets Required″ column lists the data sets that must exist to perform the specified type of restart in the specified phase.

Table 83. REORG TABLESPACE utility restart information for SHRLEVEL NONE, REFERENCE, and CHANGE

Phase	Type of restart allowed for SHRLEVEL NONE	Type of restart allowed for SHRLEVEL REFERENCE	Type of restart allowed for SHRLEVEL CHANGE	Required data sets	Notes
UNLOAD	CURRENT, PHASE	CURRENT, PHASE	None	SYSREC	
RELOAD	CURRENT, PHASE	CURRENT, PHASE	None	SYSREC	1, 2
SORT	CURRENT, PHASE	CURRENT, PHASE	None	None	2, 3
BUILD	CURRENT, PHASE	CURRENT, PHASE	None	None	2, 3, 4
SORTBLD	CURRENT, PHASE	CURRENT, PHASE	None	None	2
LOG	Phase does not occur	Phase does not occur	None	None	
SWITCH	Phase does not occur	CURRENT, PHASE	CURRENT, PHASE	Originals and shadows	3
BUILD2	Phase does not occur	CURRENT, PHASE	CURRENT, PHASE	Shadows for nonpartitioning indexes	3, 4

Notes:
1. For None, if you specify NOSYSREC, restart is not possible, and you must execute the RECOVER TABLESPACE utility for the table space or partition. For REFERENCE, if the REORG job includes both SORTDATA and NOSYSREC, RESTART or RESTART(PHASE) restarts at the beginning of the UNLOAD phase.
2. If you specify SHRLEVEL NONE or SHRLEVEL REFERENCE, and the job includes the SORTKEYS option, use RESTART or RESTART(PHASE) to restart at the beginning of the RELOAD phase.
3. You can restart the utility with RESTART or RESTART(PHASE). However, because this phase does not take checkpoints, RESTART restarts from the beginning of the phase.
4. If you specify the PART option with REORG TABLESPACE, you cannot restart the utility at the beginning of the BUILD or BUILD2 phase if any nonpartitioning index is in a page set REBUILD-pending (PSRBD) status.

If you restart a REORG STATISTICS job by using RESTART CURRENT, inline statistics are not collected. To update catalog statistics, run the RUNSTATS utility after the restarted job completes. Restarting a REORG STATISTICS job with RESTART(PHASE) is conditional after executing UNLOAD PAUSE. To determine if catalog table statistics are going to be updated, see Table 84. This table shows whether or not statistics are updated for REORG STATISTICS jobs according to the phase in which the job terminated and the restart value that was used.

Table 84. Statistics collection for REORG TABLESPACE utility phase restart

Phase	CURRENT	PHASE
UTILINIT	NO	YES
UNLOAD	NO	YES
RELOAD	NO	YES
SORT	NO	NO

Table 84. Statistics collection for REORG TABLESPACE utility phase restart (continued)

Phase	CURRENT	PHASE
BUILD	NO	YES
SORTBLD	NO	YES

For instructions on restarting a utility job, see Chapter 3, "Invoking DB2 online utilities," on page 15.

Restarting REORG after an out-of-space condition: See "Restarting after the output data set is full" on page 45 for guidance in restarting REORG from the last commit point after receiving an out-of-space condition.

Concurrency and compatibility for REORG TABLESPACE

DB2 treats individual data and index partitions, and individual logical partitions of nonpartitioning indexes as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible.

REORG of a LOB table space is not compatible with any other utility. The LOB table space is unavailable to other applications during REORG processing.

This section includes a series of tables that show which claim classes REORG drains and any restrictive state that the utility sets on the target object.

For nonpartitioned indexes, REORG PART:

- Drains only the logical partition (and the repeatable read class for the entire index)
- Does not set the page set REBUILD-pending status (PSRCP)
- Does not use PCTFREE or FREEPAGE attributes when inserting keys

For SHRLEVEL NONE, Table 85 shows which claim classes REORG drains and any restrictive state that the utility sets on the target object. For each column, the table indicates the claim or drain that is acquired and the restrictive state that is set in the corresponding phase. UNLOAD CONTINUE and UNLOAD PAUSE, unlike UNLOAD ONLY, include the RELOAD phase and thus include the drains and restrictive states of that phase.

Table 85. Claim classes of REORG TABLESPACE SHRLEVEL NONE operations

Target	UNLOAD phase of REORG	RELOAD phase of REORG if UNLOAD CONTINUE or PAUSE	UNLOAD phase of REORG PART	RELOAD phase of REORG PART if UNLOAD CONTINUE or PAUSE
Table space, partition, or a range of partitions of a table space	DW/UTRO	DA/UTUT	DW/UTRO	DA/UTUT
Partitioning index, data-partitioned secondary index, or partition of either type of index	DW/UTRO	DA/UTUT	DW/UTRO	DA/UTUT
Nonpartitioned index	DW/UTRO	DA/UTUT	None	DR

Table 85. Claim classes of REORG TABLESPACE SHRLEVEL NONE operations (continued)

Target	UNLOAD phase of REORG	RELOAD phase of REORG if UNLOAD CONTINUE or PAUSE	UNLOAD phase of REORG PART	RELOAD phase of REORG PART if UNLOAD CONTINUE or PAUSE
Logical partition of nonpartitioning index	None	None	DW/UTRO	DA/UTUT

Legend:
- DA: Drain all claim classes, no concurrent SQL access.
- DR: Drain the repeatable read class, no concurrent access for SQL repeatable readers.
- DW: Drain the write claim class, concurrent access for SQL readers.
- UTUT: Utility restrictive state, exclusive control.
- UTRO: Utility restrictive state, read-only access allowed.
- None: Any claim, drain, or restrictive state for this object does not change in this phase.

For SHRLEVEL REFERENCE, Table 86 shows which claim classes REORG drains and any restrictive state that the utility sets on the target object. For each column, the table indicates the claim or drain that is acquired and the restrictive state that is set in the corresponding phase.

Table 86. Claim classes of REORG TABLESPACE SHRLEVEL REFERENCE operations

Target	UNLOAD phase of REORG	SWITCH phase of REORG	UNLOAD phase of REORG PART	SWITCH phase of REORG PART	BUILD2 phase of REORG PART
Table space or partition of table space	DW/UTRO	DA/UTUT	DW/UTRO	DA/UTUT	UTRW
Partitioning index, data- partitioned secondary index, or partition of either	DW/UTRO	DA/UTUT	DW/UTRO	DA/UTUT	UTRW
Nonpartitioned secondary index	DW/UTRO	DA/UTUT	None	DR	None
Logical partition of nonpartitioning index	None	None	DW/UTRO	DA/UTUT	None

Legend:
- DA: Drain all claim classes, no concurrent SQL access.
- DDR: Dedrain the read claim class, concurrent SQL access.
- DR: Drain the repeatable read class, no concurrent access for SQL repeatable readers.
- DW: Drain the write claim class, concurrent access for SQL readers.
- UTUT: Utility restrictive state, exclusive control.
- UTRO: Utility restrictive state, read-only access allowed.
- None: Any claim, drain, or restrictive state for this object does not change in this phase.

For REORG of an entire table space with SHRLEVEL CHANGE, Table 87 shows which claim classes REORG drains and any restrictive state that the utility sets on the target object.

Table 87. Claim classes of REORG TABLESPACE SHRLEVEL CHANGE operations

Target	UNLOAD phase	Last iteration of LOG phase	SWITCH phase
Table space	CR/UTRW[1]	DW/UTRO	DA/UTUT

Table 87. Claim classes of REORG TABLESPACE SHRLEVEL CHANGE operations (continued)

Target	UNLOAD phase	Last iteration of LOG phase	SWITCH phase
Index	CR/UTRW[1]	DW/UTRO	DA/UTUT

Legend:
- CR: Claim the read claim class.
- DA: Claim all claim classes, no concurrent SQL access.
- DW: Drain the write claim class, concurrent access for SQL readers.
- UTUT: Utility restrictive state, exclusive control.
- UTRO: Utility restrictive state, read-only access allowed.
- UTRW: Utility restrictive state, read-write access allowed.

Notes:

1. If the target object is a segmented table space, SHRLEVEL CHANGE does not allow you to concurrently execute an SQL searched DELETE without the WHERE clause.

For REORG of a partition with SHRLEVEL CHANGE, Table 88 shows which claim classes REORG drains and any restrictive state that the utility sets on the target object.

Table 88. Claim classes of REORG TABLESPACE SHRLEVEL CHANGE operations on a partition

Target	UNLOAD phase	Last iteration of LOG phase	SWITCH phase	BUILD2 phase
Partition of table space	CR/UTRW	DW/UTRO or DA/UTUT[1]	DA/UTUT	UTRW
Partition of partitioning index	CR/UTRW	DW/UTRO or DA/UTUT[1]	DA/UTUT	UTRW
Nonpartitioning index	None	None	DR	None
Logical partition of nonpartitioning index	CR/UTRW	DW/UTRO or DA/UTUT[1]	DA/UTUT	None

Legend:
- CR: Claim the read claim class.
- DA: Drain all claim classes, no concurrent SQL access.
- DDR: Dedrain the read claim class, no concurrent access for SQL repeatable readers.
- DR: Drain the repeatable read class, no concurrent access for SQL repeatable readers.
- DW: Drain the write claim class, concurrent access for SQL readers.
- UTUT: Utility restrictive state, exclusive control.
- UTRO: Utility restrictive state, read-only access allowed.
- UTRW: Utility restrictive state, read-write access allowed.
- None: Any claim, drain, or restrictive state for this object does not change in this phase.

Notes:

1. DA/UTUT applies if you specify DRAIN ALL.

Table 89 on page 483 shows which utilities can run concurrently with REORG on the same target object. The target object can be a table space, an index space, or a partition of a table space or index space. If compatibility depends on particular options of a utility, that information is also shown.

Table 89. Compatibility of REORG TABLESPACE with other utilities

Action	REORG SHRLEVEL NONE UNLOAD CONTINUE or PAUSE, REORG SHRLEVEL REFERENCE, or REORG SHRLEVEL CHANGE	REORG SHRLEVEL NONE UNLOAD ONLY without clustering index	REORG SHRLEVEL NONE UNLOAD ONLY with clustering index
CATMAINT	No	No	No
CHECK DATA	No	No	No
CHECK INDEX	No	Yes	Yes
CHECK LOB	No	No	No
COPY INDEXSPACE	No	Yes	Yes
COPY TABLESPACE	No	Yes	Yes
DIAGNOSE	Yes	Yes	Yes
LOAD	No	No	No
MERGECOPY	No	No	No
MODIFY	No	No	No
QUIESCE	No	Yes	Yes
REBUILD INDEX	No	Yes	No
RECOVER INDEX	No	Yes	No
RECOVER INDEXSPACE	No	No	No
RECOVER TABLESPACE	No	No	No
REORG INDEX	No	Yes	No
REORG TABLESPACE SHRLEVEL NONE UNLOAD CONTINUE or PAUSE, REORG SHRLEVEL REFERENCE, or REORG SHRLEVEL CHANGE	No	No	No
REORG TABLESPACE SHRLEVEL NONE UNLOAD ONLY or EXTERNAL	No	Yes	Yes
REPAIR DUMP or VERIFY	No	Yes	Yes
REPAIR LOCATE KEY or RID DELETE or REPLACE	No	No	No
REPAIR LOCATE INDEX PAGE REPLACE	No	Yes	No
REPAIR LOCATE TABLESPACE PAGE REPLACE	No	No	No
REPORT	Yes	Yes	Yes
RUNSTATS	No	Yes	Yes
STOSPACE	No	Yes	Yes
UNLOAD	No	Yes	Yes

Table 90 on page 484 shows which DB2 operations can be affected when reorganizing catalog table spaces.

Table 90. DB2 operations that are affected by reorganizing catalog table spaces

Catalog table space	Actions that might not run concurrently
Any table space except SYSCOPY and SYSSTR	CREATE, ALTER, and DROP statements
SYSCOPY, SYSDBASE, SYSDBAUT, SYSSTATS, SYSUSER, SYSHIST	Utilities
SYSDBASE, SYSDBAUT, SYSGPAUT, SYSPKAGE, SYSPLAN, SYSUSER	GRANT and REVOKE statements
SYSDBAUT, SYSDBASE, SYSGPAUT, SYSPKAGE, SYSPLAN, SYSSTATS, SYSUSER, SYSVIEWS	BIND and FREE commands
SYSPKAGE, SYSPLAN	Plan or package execution

Reviewing REORG TABLESPACE output

The output from REORG TABLESPACE consists of a reorganized table space, partition, or a range of partitions; from REORG INDEX, the output consists of a reorganized index or index partition. Table 91 summarizes the effect of REORG on a table space partition and on the corresponding index partition.

Table 91. Summary of the results of REORG TABLESPACE according to the type of specification.

Specification	Results
REORG TABLESPACE	All data + entire partitioning index + all nonpartitioning indexes
REORG TABLESPACE PART *n*	Data for PART *n* + PART *n* of the partitioning index + index entries for PART *n* in all nonpartitioning indexes
REORG TABLESPACE PART *n1*:*n2*	Data for PART *n1* through *n2* + PART *n1* through *n2* of the partitioning index + index entries for those partitions in all nonpartitioning indexes
REORG TABLESPACE SCOPE PENDING	Specified table space or partitions that are in REORG-pending status.

When reorganizing a segmented table space, REORG leaves free pages and free space on each page in accordance with the current values of the FREEPAGE and PCTFREE parameters. (You can set those values by using the CREATE TABLESPACE, ALTER TABLESPACE, CREATE INDEX, or ALTER INDEX statements). REORG leaves one free page after reaching the FREEPAGE limit for each table in the table space. When reorganizing a nonsegmented table space, REORG leaves one free page after reaching the FREEPAGE limit, regardless of whether the loaded records belong to the same or different tables.

After running REORG TABLESPACE

After a reorganization is complete, perform the following actions:

- If you have used LOG YES, consider taking an image copy of the reorganized table space or partition to:
  - Provide a full image copy for recovery. This action prevents the need to process the log records that are written during reorganization.
  - Permit making incremental image copies later.

You might not need to take an image copy of a table space for which all the following statements are true:
- The table space is relatively small.
- The table space is used only in read-only applications.
- The table space can be easily loaded again in the event of failure.

See Chapter 11, "COPY," on page 103 for information about making image copies.

- If you use REORG SHRLEVEL NONE LOG NO on a LOB table space and DB2 determines that nothing needs to be done to the table space, no COPY-pending status is set. However, if DB2 indicates that changes are needed, REORG places the reorganized LOB table space or partition in COPY-pending status. In this situation, perform a full image copy to reset the COPY-pending status and to ensure that a backup is available for recovery.

  You should also run the COPY utility if the REORG was performed to turn off REORG-pending status (REORP), and an inline copy was not taken. You cannot use an image copy that was created before turning off REORP.
- If you use COPYDDN, SHRLEVEL REFERENCE, or SHRLEVEL CHANGE, and the object that you are reorganizing is not a catalog or directory table space for which COPYDDN is ignored, you do not need to take an image copy.
- Use the RUNSTATS utility on the table space and its indexes if inline statistics were not collected, so that the DB2 catalog statistics take into account the newly reorganized data, and SQL paths can be selected with accurate information. You need to run RUNSTATS on nonpartitioning indexes only if you reorganized a subset of the partitions.
- If you use REORG TABLESPACE SHRLEVEL CHANGE, you can drop the mapping table and its index.
- If you use SHRLEVEL REFERENCE or CHANGE, and a table space, partition, or index resides in user-managed data sets, you can delete the user-managed shadow data sets.
- If you specify DISCARD on a REORG of a table that is involved in a referential integrity set, you need to run CHECK DATA for any affected referentially related objects that were placed in CHECK-pending status.

Effects of running REORG TABLESPACE

This section contains information about the effects of running REORG TABLESPACE.

The effect of REORG TABLESPACE on index version numbers and the version of the data

DB2 stores the range of used version numbers in the OLDEST_VERSION and CURRENT_VERSION columns of one or more of the following catalog tables, depending on the object:
- SYSIBM.SYSTABLESPACE
- SYSIBM.SYSTABLESPART
- SYSIBM.SYSINDEXES
- SYSIBM.SYSINDEXPART

The OLDEST_VERSION column contains the oldest used version number, and the CURRENT_VERSION column contains the current version number.

When you run REORG TABLESPACE, the utility sets all of the rows in the table or partition to the current object version. The utility also updates the range of used version numbers for indexes that are defined with the COPY NO attribute. REORG

TABLESPACE sets the OLDEST_VERSION column equal to the CURRENT_VERSION column in the appropriate catalog column. These updated values indicate that only one version is active. DB2 can then reuse all of the other version numbers.

Recycling of version numbers is required when all of the version numbers are being used. All version numbers are being used when one of the following situations is true:

- The value in the CURRENT_VERSION column is one less than the value in the OLDEST_VERSION column.
- The value in the CURRENT_VERSION column is 255 for table spaces or 15 for indexes, and the value in the OLDEST_VERSION column is 0 or 1.

You can also run LOAD REPLACE, REBUILD INDEX, or REORG INDEX to recycle version numbers for indexes that are defined with the COPY NO attribute. To recycle version numbers for indexes that are defined with the COPY YES attribute or for table spaces, run MODIFY RECOVERY.

For more information about versions and how they are used by DB2, see Part 2 of *DB2 Administration Guide*.

The effect of REORG TABLESPACE on the control interval

When you run REORG TABLESPACE without the REUSE option and the data set that contains that data is DB2–managed, DB2 deletes this data set before the REORG and redefines a new data set with a control interval that matches the page size.

Sample REORG TABLESPACE control statements

Example 1: Reorganizing a table space. The following control statement specifies that the REORG TABLESPACE utility is to reorganize table space DSN8S81D in database DSN8D81A.

```
REORG TABLESPACE DSN8D81A.DSN8S81D
```

Example 2: Reorganizing a table space and specifying the unload data set. The control statement in Figure 79 specifies that REORG TABLESPACE is to reorganize table space DSN8D81A.DSN8S81D. The DD name for the unload data set is UNLD, as specified by the UNLDDN option.

```
//STEP1    EXEC DSNUPROC,UID='IUJLU101.REORG',
//         UTPROC='',
//         SYSTEM='DSN'
//UTPRINT  DD  SYSOUT=*
//UNLD     DD DSN=IUJLU101.REORG.STEP1.UNLD,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTWK01  DD DSN=IUJLU101.REORG.STEP1.SORTWK01,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTWK02  DD DSN=IUJLU101.REORG.STEP1.SORTWK02,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
REORG TABLESPACE (DSN8D81A.DSN8S81D)
      UNLDDN (UNLD)
//*
```

Figure 79. Example REORG TABLESPACE control statement with the UNLDDN option

Example 3: Reorganizing a table space partition. The following control statement specifies that REORG TABLESPACE is to reorganize partition 3 of table space

DSN8D81A.DSN8S81E. The SORTDEVT option indicates the device type for the temporary data sets that are to be dynamically allocated by DFSORT.

```
REORG TABLESPACE DSN8D81A.DSN8S81E
  PART 3
  SORTDEVT SYSDA
```

Example 4: Reorganizing a table and using parallel index build. The control statement in Figure 80 specifies that REORG TABLESPACE is to reorganize table space DSNDB04.DSN8S81D and to use a parallel index build to rebuild the indexes. The indexes are built in parallel, because more than one index needs to be built and the job allocates the data sets that DFSORT needs. Note that you no longer need to specify SORTKEYS; it is the default.

The job allocates the sort work data sets in two groups, which limits the number of pairs of utility subtasks to two. This example does not require UTPRIN*nn* DD statements because it uses DSNUPROC to invoke utility processing. DSNUPROC includes a DD statement that allocates UTPRINT to SYSOUT.

LOG NO specifies that records are not to be logged during the RELOAD phase. This option puts the table space in COPY-pending status.

```
//SAMPJOB  JOB  ...
//STEP1    EXEC DSNUPROC,UID='SAMPJOB.REORG',UTPROC='',SYSTEM='DSN'
//SYSREC   DD DSN=SAMPJOB.REORG.STEP1.SYSREC,DISP=(NEW,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//* First group of sort work data sets for parallel index build
//SW01WK01 DD UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SW01WK02 DD UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SW01WK03 DD UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//* Second group of sort work data sets for parallel index build
//SW02WK01 DD UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SW02WK02 DD UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SW02WK03 DD UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//* Sort work data sets for use by SORTDATA
//SORTWK01 DD  UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SORTWK02 DD  UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SORTWK03 DD  UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SYSIN    DD *
REORG TABLESPACE DSNDB04.DSN8S81D LOG NO
/*
```

Figure 80. Example REORG TABLESPACE control statement with LOG NO option

Example 5: Reorganizing a table while allowing read-write access. The following control statement specifies that REORG TABLESPACE is to reorganize table space DSNDB04.DSN8S81E and to use a parallel index build to rebuild the indexes. DFSORT dynamically allocates sort work data sets. This example does not require UTPRIN*nn* DD statements because it uses DSNUPROC to invoke utility processing. DSNUPROC includes a DD statement that allocates UTPRINT to SYSOUT. The SORTDEVT option indicates the device type for the temporary data sets that are to be dynamically allocated by DFSORT. The SHRLEVEL CHANGE option specifies that while the table is being reorganized, users have read-write access. The name of the mapping table is DSN8MAP. This table is used to map the RIDs of data records in the original copy of the area to the corresponding RIDs in the shadow copy.

```
//SAMPJOB  JOB  ...
//STEP1    EXEC DSNUPROC,UID='SAMPJOB.REORG',UTPROC='',SYSTEM='DSN'
//SYSCOPY  DD UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND),
//            DSN=SAMPJOB,COPY,DISP=(NEW,CATLG,CATLG)
```

```
//SYSIN    DD *
REORG TABLESPACE DSNDB04.DSN8S81E LOG NO SORTDEVT SYSDA SORTNUM 4
  SHRLEVEL CHANGE MAPPINGTABLE DSN8MAP
/*
```

Example 6: Specifying a deadline for the SWITCH phase while reorganizing a table. The following control statement specifies that REORG TABLESPACE is to reorganize table space DSN8D81A.DSN8S81D. The DEADLINE option indicates that the deadline for start of the SWITCH phase is eight hours from the start of the REORG job. The COPYDDN and RECOVERYDDN options indicate that the utility is to take an image copy of the table space. DB2 is to write the primary image copy at the local site to a data set that is defined by the MYCOPY1 DD statement and to write the primary image copy at the recovery site to a data set that is defined by the MYCOPY2 DD statement. SHRLEVEL REFERENCE indicates that access is restricted during reorganization.

```
REORG TABLESPACE DSN8D81A.DSN8S81D COPYDDN(MYCOPY1)
  RECOVERYDDN(MYCOPY2) SHRLEVEL REFERENCE
  DEADLINE CURRENT TIMESTAMP + 8 HOURS
```

Example 7: Setting a deadline for a REORG TABLESPACE job. The following control statement specifies that REORG TABLESPACE is to reorganize table space DSN8D81A.DSN8S81D. The DEADLINE option indicates that the deadline for the start of the SWITCH phase is eight hours from the start of the REORG job. The name of the mapping table is DSN8810.MAP_TBL. The maximum desired amount of time for the log processing in the read-only (last) iteration of log processing is 240 seconds, as indicated by the MAXRO option. If DB2 is not reading the log quickly enough after the applications write to the log, DB2 drains the write claim class after sending the LONGLOG message to the operator. That draining takes place at least 900 seconds after the LONGLOG message is sent, as indicated by the DELAY option. DB2 is also to take inline image copies for the local site and recovery site, as indicated by the COPYDDN and RECOVERYDDN options.

```
REORG TABLESPACE DSN8D81A.DSN8S81D COPYDDN(MYCOPY1)
  RECOVERYDDN(MYCOPY2) SHRLEVEL CHANGE
  DEADLINE CURRENT TIMESTAMP + 8 HOURS
  MAPPINGTABLE DSN8810.MAP_TBL MAXRO 240 LONGLOG DRAIN DELAY 900
```

Example 8: Reorganizing a range of table space partitions. The following control statement specifies that REORG TABLESPACE is to reorganize partitions 3 through 5 of table space DSN8D81A.DSN8S81E. The SORTDEVT option indicates the device type for the temporary data sets that are to be dynamically allocated by DFSORT. The SHRLEVEL NONE option indicates that while the data is being unloaded, applications can read but can't write. While the data is being reloaded, applications can have read-write access. SHRLEVEL NONE is the default. The COPYDDN option indicates that the utility is to take an image copy of the table space and to write the primary image copy to the data set that is defined by the SYSCOPY DD statement.

```
REORG TABLESPACE DSN8D81A.DSN8S81E
  PART 3:5
    SORTDEVT SYSDA
    SHRLEVEL NONE
    COPYDDN SYSCOPY
```

Example 9: Reorganizing a partition and updating the statistics. The following control statement specifies that REORG TABLESPACE is to reorganize partition 3 of table space DSN8D81A. DSN8S81E. The STATISTICS option indicates that the utility is also to update statistics in the catalog for that partition. Note that the STATISTICS option is not valid for LOB table spaces.

```
REORG TABLESPACE DSN8D81A.DSN8S81E
   STATISTICS PART 3
```

Example 10: Reorganizing a table space and reporting table space and index statistics. The following control statement specifies that REORG TABLESPACE is to reorganize table space DSN8D81A.DSN8S81E. The SORTDATA option indicates that the data is to be unloaded and sorted in clustering order. This option is the default and does not need to be specified. The STATISTICS, TABLE, INDEX, and REPORT YES options indicate that the utility is also to report catalog statistics for all tables in the table space and for all indexes that are defined on those tables. The KEYCARD, FREQVAL, NUMCOLS, and COUNT options indicate that DB2 is to collect 10 frequent values on the first key column of the index. UPDATE NONE indicates that the catalog tables are not to be updated. This option requires that REPORT YES also be specified.

```
REORG TABLESPACE DSN8D81A.DSN8S81E SORTDATA STATISTICS
   TABLE
   INDEX(ALL) KEYCARD FREQVAL NUMCOLS 1
   COUNT 10 REPORT YES UPDATE NONE
```

Example 11: Determining whether a table space should be reorganized. The control statement in Figure 81 specifies that REORG TABLESPACE is to report if the OFFPOSLIMIT and INDREFLIMIT values for partition 11 of table space DBHR5201.TPHR5201 exceed the specified values (11 for OFFPOSLIMIT and 15 for INDREFLIMIT).

```
//STEP1    EXEC DSNUPROC,UID='HUHRU252.REORG2',TIME=1440,
//         UTPROC='',
//         SYSTEM='DSN',DB2LEV=DB2A
//SYSREC   DD DSN=HUHRU252.REORG2.STEP1.SYSREC,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSCOPY  DD DSN=HUHRU252.REORG2.STEP1.SYSCOPY,DISP=(MOD,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//         SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
REORG TABLESPACE DBHR5201.TPHR5201 PART 11
          NOSYSREC
          REPORTONLY
          SHRLEVEL CHANGE MAPPINGTABLE ADMF001.MAP1
          COPYDDN (SYSCOPY)
          OFFPOSLIMIT 11 INDREFLIMIT 15
/*
```

Figure 81. Example REORG TABLESPACE statement with REPORTONLY, OFFPOSLIMIT, and INDREFLIMIT options

On successful completion, DB2 returns output that is similar to the output in Figure 82 on page 490. This sample output shows that the limits have been met.

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = HUHRU252.REORG2
DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    DSNUGUTC -  REORG TABLESPACE DBHR5201.TPHR5201 PART 11 NOSYSREC REPORTONLY SHRLEVEL CHANGE
MAPPINGTABLE ADMF001.MAP1 COPYDDN(SYSCOPY) OFFPOSLIMIT 11 INDREFLIMIT 15
DSNU286I  = DSNURLIM - REORG TABLESPACE DBHR5201.TPHR5201 OFFPOSLIMIT SYSINDEXPART ROWS
* CREATOR.IXNAME : ADMF001.IPHR5201
  CREATOR.TBNAME : ADMF001.TBHR5201
  PART:    1 CARDF:  6.758E+03   FAROFFPOSF:   2.892E+03   NEAROFFPOSF:  8.18E+02    STATSTIME: 2003-04-11
13.32.06
DSNU287I  = DSNURLIM - REORG TABLESPACE DBHR5201.TPHR5201 INDREFLIMIT SYSTABLEPART ROWS
  DBNAME  .TSNAME   PART          CARD  FARINDREF NEARINDREF      STATSTIME
  DBHR5201.TPHR5201     1         6758          0          0 2003-04-11-13.32.06

DSNU289I  = DSNURLIM -  REORG LIMITS HAVE BEEN MET
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 82. Sample output showing that REORG limits have been met

Example 12: Conditionally reorganizing a table space. In the example in Figure 83, the RUNSTATS utility control statement specifies that the utility is to update space statistics in the catalog for table space DBHR5201.TPHR5201. This RUNSTATS job ensures that the space statistics for this table space are current. The subsequent REORG TABLESPACE control statement specifies that if any of the values for OFFPOSLIMIT or INDREFLIMIT exceed 9, the utility is to reorganize the table space.

```
//*****************************************************************
//* COMMENT: UPDATE STATISTICS
//*****************************************************************
//STEP1    EXEC DSNUPROC,UID='HUHRU252.REORG1',TIME=1440,
//         UTPROC='',
//         SYSTEM='DSN',DB2LEV=DB2A
//SYSREC   DD DSN=HUHRU252.REORG1.STEP1.SYSREC,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//         SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
RUNSTATS TABLESPACE DBHR5201.TPHR5201
      UPDATE SPACE
/*
//*****************************************************************
//* COMMENT: REORG THE TABLESPACE
//*****************************************************************
//STEP2    EXEC DSNUPROC,UID='HUHRU252.REORG1',TIME=1440,
//         UTPROC='',
//         SYSTEM='DSN',DB2LEV=DB2A
//SYSREC   DD DSN=HUHRU252.REORG1.STEP1.SYSREC,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSCOPY1 DD DSN=HUHRU252.REORG1.STEP1.SYSCOPY1,
//         DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//         SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
REORG TABLESPACE DBHR5201.TPHR5201
      SHRLEVEL CHANGE MAPPINGTABLE MAP1
      COPYDDN(SYSCOPY1)
      OFFPOSLIMIT 9 INDREFLIMIT 9
/*
```

Figure 83. Example of conditionally reorganizing a table

On successful completion, DB2 returns output for the REORG TABLESPACE job that is similar to the output in Figure 84 on page 491.

```
DSNU050I    DSNUGUTC -  REORG TABLESPACE DBHR5201.TPHR5201 SHRLEVEL CHANGE MAPPINGTABLE
MAP1 COPYDDN(SYSCOPY1)
OFFPOSLIMIT 9 INDREFLIMIT 9
DSNU286I  = DSNURLIM - REORG TABLESPACE DBHR5201.TPHR5201 OFFPOSLIMIT SYSINDEXPART ROWS
* CREATOR.IXNAME : ADMF001.IPHR5201
CREATOR.TBNAME : ADMF001.TBHR5201
   PART:   1 CARDF:  3.6E+01     FAROFFPOSF:   0.0E0       NEAROFFPOSF:  1.2E+01
STATSTIME: 2002-05-28-16.22.18
  CREATOR.IXNAME : ADMF001.IPHR5201
  CREATOR.TBNAME : ADMF001.TBHR5201
   PART:   2 CARDF:  5.0E+00     FAROFFPOSF:   0.0E0       NEAROFFPOSF:  0.0E0
STATSTIME: 2002-05-28-16.22.18
...
* CREATOR.IXNAME : ADMF001.IPHR5201
  CREATOR.TBNAME : ADMF001.TBHR5201
   PART:  11 CARDF:  6.758E+03   FAROFFPOSF:   2.892E+03   NEAROFFPOSF:  8.18E+02
STATSTIME: 2002-05-28-16.22.18
DSNU287I  = DSNURLIM - REORG TABLESPACE DBHR5201.TPHR5201 INDREFLIMIT SYSTABLEPART ROWS
  DBNAME  .TSNAME   PART         CARD  FARINDREF NEARINDREF      STATSTIME
  DBHR5201.TPHR5201    1           36          0          0 2002-05-28-16.22.18
  DBHR5201.TPHR5201    2            5          0          0 2002-05-28-16.22.18
  DBHR5201.TPHR5201    3           54          0          0 2002-05-28-16.22.18
  DBHR5201.TPHR5201    4           30          0          0 2002-05-28-16.22.18
  DBHR5201.TPHR5201    5           21          0          0 2002-05-28-16.22.18
  DBHR5201.TPHR5201    6            5          0          0 2002-05-28-16.22.18
  DBHR5201.TPHR5201    7            4          0          0 2002-05-28-16.22.18
  DBHR5201.TPHR5201    8           35          0          0 2002-05-28-16.22.18
  DBHR5201.TPHR5201    9           25          0          0 2002-05-28-16.22.18
  DBHR5201.TPHR5201   10            1          0          0 2002-05-28-16.22.18
  DBHR5201.TPHR5201   11         6758          0          0 2002-05-28-16.22.18
DSNU289I  = DSNURLIM -  REORG LIMITS HAVE BEEN MET
DSNU290I  = DSNURLIM -  REORG WILL BE PERFORMED
DSNU252I    DSNUGSRT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=6985 FOR
TABLESPACE DBHR5201.TPHR5201
DSNU250I    DSNUGSRT - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:01
DSNU304I  = DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=6985 FOR TABLE
ADMF001.TBHR5201
DSNU302I    DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=6985
DSNU300I    DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:29
DSNU042I    DSNUGSOR - SORT PHASE STATISTICS -
                       NUMBER OF RECORDS=34925
                       ELAPSED TIME=00:00:00

DSNU348I  = DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=36 FOR INDEX ADMF001.IPHR5201  PART 1
DSNU348I  = DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=5 FOR INDEX ADMF001.IPHR5201  PART 2
...
DSNU349I  = DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=6985 FOR INDEX ADMF001.IUHR5210
DSNU258I    DSNURBXD - BUILD PHASE STATISTICS - NUMBER OF INDEXES=5
DSNU259I    DSNURBXD - BUILD PHASE COMPLETE, ELAPSED TIME=00:00:18
DSNU386I    DSNURLGD - LOG PHASE STATISTICS. NUMBER OF ITERATIONS = 1, NUMBER OF LOG
RECORDS = 194
DSNU385I    DSNURLGD - LOG PHASE COMPLETE, ELAPSED TIME = 00:01:10
DSNU400I    DSNURBID - COPY PROCESSED FOR TABLESPACE DBHR5201.TPHR5201
                       NUMBER OF PAGES=1073
                       AVERAGE PERCENT FREE SPACE PER PAGE = 14.72
                       PERCENT OF CHANGED PAGES =100.00
                       ELAPSED TIME=00:01:58
DSNU387I    DSNURSWT - SWITCH PHASE COMPLETE, ELAPSED TIME = 00:01:05
DSNU428I    DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DBHR5201.TPHR5201
```

Figure 84. Sample REORG output for conditional REORG

Example 13: Reorganizing a table space after waiting for SQL statements to complete. The control statement in Figure 85 on page 492 specifies that REORG TABLESPACE is to reorganize the table space in the REORG_TBSP list, which is defined in the preceding LISTDEF utility control statement. Before reorganizing the

table space, REORG TABLESPACE is to wait for 30 seconds for SQL statements to finish adding or changing data. This interval is indicated by the DRAIN_WAIT option. If the SQL statements do not finish, the utility is to retry up to four times, as indicated by the RETRY option. The utility is to wait 10 seconds between retries, as indicated by the RETRY_DELAY option.

The TEMPLATE utility control statements define the data set characteristics for the data sets that are to be dynamically allocated during the REORG TABLESPACE job. The OPTIONS utility control statement indicates that the TEMPLATE statements and LISTDEF statement are to run in PREVIEW mode.

```
//STEP1    EXEC DSNUPROC,UID='HUHRU257.REORG',TIME=1440,
//         UTPROC='',
//         SYSTEM='DSN',DB2LEV=DB2A
//UTPRINT  DD  SYSOUT=*
//SYSIN    DD *
   OPTIONS PREVIEW
   TEMPLATE CPYTMP UNIT(SYSDA)
                   DSN(HUHRU257.REORG.T&TI..SYSCOPY1)
   TEMPLATE SREC
            UNIT(SYSDA) DISP(NEW,CATLG,CATLG)
            DSN(HUHRU257.REORG.&ST..SREC)
   TEMPLATE SDISC
            UNIT(SYSDA) DISP(NEW,CATLG,CATLG)
            DSN(HUHRU257.REORG.&ST..SDISC)
   TEMPLATE SPUNCH
            UNIT(SYSDA) DISP(NEW,CATLG,CATLG)
                 DSN(HUHRU257.REORG.&ST..SPUNCH)
   LISTDEF REORG_TBSP INCLUDE TABLESPACE DBHR5701.TPHR5701
   OPTIONS OFF
   REORG TABLESPACE LIST REORG_TBSP
       DRAIN_WAIT 30    RETRY 4     RETRY_DELAY 10
          STATISTICS
          TABLE (ALL) SAMPLE 60
          INDEX (ALL KEYCARD FREQVAL NUMCOLS 2 COUNT 15)
          SHRLEVEL CHANGE MAPPINGTABLE MAP5702
          LONGLOG DRAIN MAXRO DEFER DELAY 30
          COPYDDN (CPYTMP)
          SORTDEVT SYSDA SORTNUM 8
          PUNCHDDN SPUNCH
          DISCARDDN SDISC
                 UNLDDN SREC
```

Figure 85. Example of reorganizing a table space by using DRAIN WAIT, RETRY, and RETRY_DELAY

On successful completion, DB2 returns output similar to the output in Figure 86 on page 493.

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = HUHRU257.REORG
DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    DSNUGUTC -  OPTIONS PREVIEW
DSNU1000I   DSNUGUTC - PROCESSING CONTROL STATEMENTS IN PREVIEW MODE
DSNU1035I   DSNUILDR - OPTIONS STATEMENT PROCESSED SUCCESSFULLY
DSNU050I    DSNUGUTC -  TEMPLATE CPYTMP UNIT(SYSDA) DSN(HUHRU257.REORG.STEP1.SYSCOPY1)
DSNU1035I   DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I    DSNUGUTC -  TEMPLATE SREC UNIT(SYSDA) DISP(NEW,CATLG,CATLG) DSN(HUHRU257.REORG.&ST..SREC)
DSNU1035I   DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I    DSNUGUTC -  TEMPLATE SDISC UNIT(SYSDA) DISP(NEW,CATLG,CATLG) DSN(HUHRU257.REORG.&ST..SDISC)
DSNU1035I   DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I    DSNUGUTC -  TEMPLATE SPUNCH UNIT(SYSDA) DISP(NEW,CATLG,CATLG) DSN(HUHRU257.REORG.&ST..SPUNCH)
DSNU1035I   DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I    DSNUGUTC -  LISTDEF REORG_TBSP INCLUDE TABLESPACE DBHR5701.TPHR5701
DSNU1035I   DSNUILDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
DSNU1020I = DSNUILSA - EXPANDING LISTDEF REORG_TBSP
DSNU1021I = DSNUILSA -   PROCESSING INCLUDE CLAUSE TABLESPACE DBHR5701.TPHR5701
DSNU1022I = DSNUILSA - CLAUSE IDENTIFIES 1 OBJECTS
DSNU1023I = DSNUILSA - LISTDEF REORG_TBSP CONTAINS 1 OBJECTS
DSNU1010I   DSNUGPVV - LISTDEF REORG_TBSP EXPANDS TO THE FOLLOWING OBJECTS:
               LISTDEF REORG_TBSP -- 00000001 OBJECTS
                 INCLUDE TABLESPACE DBHR5701.TPHR5701
DSNU050I    DSNUGUTC -  OPTIONS OFF
DSNU1035I   DSNUILDR - OPTIONS STATEMENT PROCESSED SUCCESSFULLY
DSNU050I    DSNUGUTC -  REORG TABLESPACE LIST REORG_TBSP DRAIN_WAIT 30 RETRY 4 RETRY_DELAY 10 STATISTICS
TABLE(ALL)
SAMPLE 60 INDEX(ALL KEYCARD FREQVAL NUMCOLS 2 COUNT 15) SHRLEVEL CHANGE MAPPINGTABLE MAP5702 LONGLOG DRAIN
MAXRO
DEFER DELAY 30 COPYDDN(CPYTMP) SORTDEVT SYSDA SORTNUM 8 PUNCHDDN SPUNCH DISCARDDN
SDISC UNLDDN SREC
DSNU1033I   DSNUGULM - PROCESSING LIST ITEM: TABLESPACE DBHR5701.TPHR5701
DSNU1038I   DSNUGDYN - DATA SET ALLOCATED.  TEMPLATE=CPYTMP
                       DDNAME=SYS00001
                       DSN=HUHRU257.REORG.STEP1.SYSCOPY1
DSNU252I    DSNUGSRT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=331 FOR TABLESPACE DBHR
DSNU250I    DSNUGSRT - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
DSNU395I    DSNURPIB - INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = 9
DSNU397I    DSNURPIB - NUMBER OF TASKS CONSTRAINED BY VIRTUAL STORAGE
DSNU610I  = DSNUSUTP -   SYSTABLEPART CATALOG UPDATE FOR DBHR5701.TPHR5701 SUCCESSFUL
DSNU610I  = DSNUSUPT -       SYSTABSTATS CATALOG UPDATE FOR ADMF001.TBHR5701 SUCCESSFUL
DSNU610I  = DSNUSUPC -         SYSCOLSTATS CATALOG UPDATE FOR ADMF001.TBHR5701 SUCCESSFUL
DSNU610I  = DSNUSUTB -     SYSTABLES CATALOG UPDATE FOR ADMF001.TBHR5701 SUCCESSFUL
DSNU610I  = DSNUSUCO -       SYSCOLUMNS CATALOG UPDATE FOR ADMF001.TBHR5701 SUCCESSFUL
DSNU610I  = DSNUSUTS -   SYSTABLESPACE CATALOG UPDATE FOR DBHR5701.TPHR5701 SUCCESSFUL
DSNU620I  = DSNURDRT - RUNSTATS CATALOG TIMESTAMP = 2002-08-05-16.25.20.438798
DSNU304I  = DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=331 FOR TABLE ADMF001.TBHR5701
DSNU302I    DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=331
DSNU300I    DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:14
DSNU394I  = DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=331 FOR INDEX ADMF001.IXHR5703
DSNU394I  = DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=331 FOR INDEX ADMF001.IXHR5702
```

Figure 86. Sample output of REORG TABLESPACE job with DRAIN WAIT, RETRY, and RETRY_DELAY options (Part 1 of 2)

```
DSNU394I  = DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=331 FOR INDEX ADMF001.IXHR5706
DSNU394I  = DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=331 FOR INDEX ADMF001.IXHR5705
DSNU610I  = DSNUSUIP -   SYSINDEXPART CATALOG UPDATE FOR ADMF001.IXHR5702 SUCCESSFUL
DSNU610I  = DSNUSUIX -     SYSINDEXES CATALOG UPDATE FOR ADMF001.IXHR5702 SUCCESSFUL
DSNU610I  = DSNUSUCO -       SYSCOLUMNS CATALOG UPDATE FOR ADMF001.TBHR5701 SUCCESSFUL
DSNU610I  = DSNUSUCD -       SYSCOLDIST CATALOG UPDATE FOR ADMF001.IXHR5702 SUCCESSFUL
DSNU610I  = DSNUSUIP -   SYSINDEXPART CATALOG UPDATE FOR ADMF001.IXHR5705 SUCCESSFUL
DSNU610I  = DSNUSUIX -     SYSINDEXES CATALOG UPDATE FOR ADMF001.IXHR5705 SUCCESSFUL
DSNU610I  = DSNUSUCO -       SYSCOLUMNS CATALOG UPDATE FOR ADMF001.TBHR5701 SUCCESSFUL
DSNU610I  = DSNUSUCD -       SYSCOLDIST CATALOG UPDATE FOR ADMF001.IXHR5705 SUCCESSFUL
DSNU620I  = DSNURDRI - RUNSTATS CATALOG TIMESTAMP = 2002-08-05-16.25.21.292235
DSNU610I  = DSNUSUIP -   SYSINDEXPART CATALOG UPDATE FOR ADMF001.IXHR5703 SUCCESSFUL
DSNU610I  = DSNUSUIX -     SYSINDEXES CATALOG UPDATE FOR ADMF001.IXHR5703 SUCCESSFUL
DSNU610I  = DSNUSUCO -       SYSCOLUMNS CATALOG UPDATE FOR ADMF001.TBHR5701 SUCCESSFUL
DSNU610I  = DSNUSUCD -       SYSCOLDIST CATALOG UPDATE FOR ADMF001.IXHR5703 SUCCESSFUL
DSNU610I  = DSNUSUIP -   SYSINDEXPART CATALOG UPDATE FOR ADMF001.IXHR5706 SUCCESSFUL
DSNU610I  = DSNUSUIX -     SYSINDEXES CATALOG UPDATE FOR ADMF001.IXHR5706 SUCCESSFUL
DSNU610I  = DSNUSUCO -       SYSCOLUMNS CATALOG UPDATE FOR ADMF001.TBHR5701 SUCCESSFUL
DSNU610I  = DSNUSUCD -       SYSCOLDIST CATALOG UPDATE FOR ADMF001.IXHR5706 SUCCESSFUL
DSNU620I  = DSNURDRI - RUNSTATS CATALOG TIMESTAMP = 2002-08-05-16.25.22.288665
DSNU393I  = DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=331 FOR INDEX ADMF001.IPHR5701  PART 11
DSNU394I  = DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=331 FOR INDEX ADMF001.IPHR5701
DSNU394I  = DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=331 FOR INDEX ADMF001.IXHR5704
DSNU610I  = DSNUSUIP -   SYSINDEXPART CATALOG UPDATE FOR ADMF001.IPHR5701 SUCCESSFUL
DSNU610I  = DSNUSUPI -       SYSINDEXSTATS CATALOG UPDATE FOR ADMF001.IPHR5701 SUCCESSFUL
DSNU610I  = DSNUSUPD -         SYSCOLDISTSTATS CATALOG UPDATE FOR ADMF001.IPHR5701 SUCCESSFUL
DSNU610I  = DSNUSUPC -         SYSCOLSTATS CATALOG UPDATE FOR ADMF001.TBHR5701 SUCCESSFUL
DSNU610I  = DSNUSUIX -     SYSINDEXES CATALOG UPDATE FOR ADMF001.IPHR5701 SUCCESSFUL
DSNU610I  = DSNUSUCO -       SYSCOLUMNS CATALOG UPDATE FOR ADMF001.TBHR5701 SUCCESSFUL
DSNU610I  = DSNUSUCD -       SYSCOLDIST CATALOG UPDATE FOR ADMF001.IPHR5701 SUCCESSFUL
DSNU610I  = DSNUSUIP -   SYSINDEXPART CATALOG UPDATE FOR ADMF001.IXHR5704 SUCCESSFUL
DSNU610I  = DSNUSUIX -     SYSINDEXES CATALOG UPDATE FOR ADMF001.IXHR5704 SUCCESSFUL
DSNU610I  = DSNUSUCO -       SYSCOLUMNS CATALOG UPDATE FOR ADMF001.TBHR5701 SUCCESSFUL
DSNU610I  = DSNUSUCD -       SYSCOLDIST CATALOG UPDATE FOR ADMF001.IXHR5704 SUCCESSFUL
DSNU620I  = DSNURDRI - RUNSTATS CATALOG TIMESTAMP = 2002-08-05-16.25.20.886803
DSNU391I    DSNURPTB - SORTBLD PHASE STATISTICS. NUMBER OF INDEXES = 7
DSNU392I    DSNURPTB - SORTBLD PHASE COMPLETE, ELAPSED TIME = 00:00:04
DSNU377I  = DSNURLOG - IN REORG WITH SHRLEVEL CHANGE, THE LOG IS
BECOMING LONG, MEMBER=        , UTILID=HUHRU257.REORG
DSNU377I  = DSNURLOG - IN REORG WITH SHRLEVEL CHANGE, THE LOG IS
BECOMING LONG, MEMBER=        , UTILID=HUHRU257.REORG
...
DSNU377I  = DSNURLOG - IN REORG WITH SHRLEVEL CHANGE, THE LOG IS
BECOMING LONG, MEMBER=        , UTILID=HUHRU257.REORG
DSNU1122I = DSNURLOG - JOB T3161108 PERFORMING REORG
WITH UTILID HUHRU257.REORG UNABLE TO DRAIN DBHR5701.TPHR5701.
RETRY 1 OF 4 WILL BE ATTEMPTED IN 10 SECONDS
 DSNU1122I = DSNURLOG - JOB T3161108 PERFORMING REORG
 WITH UTILID HUHRU257.REORG UNABLE TO DRAIN DBHR5701.TPHR5701.
RETRY 2 OF 4 WILL BE ATTEMPTED IN 10 SECONDS
DSNU386I    DSNURLGD - LOG PHASE STATISTICS. NUMBER OF ITERATIONS = 32, NUMBER OF LOG RECORDS = 2288
DSNU385I    DSNURLGD - LOG PHASE COMPLETE, ELAPSED TIME = 00:03:43
DSNU400I    DSNURBID - COPY PROCESSED FOR TABLESPACE DBHR5701.TPHR5701
                       NUMBER OF PAGES=377
                      AVERAGE PERCENT FREE SPACE PER PAGE =  5.42
                       PERCENT OF CHANGED PAGES =100.00
                       ELAPSED TIME=00:04:02
DSNU387I    DSNURSWT - SWITCH PHASE COMPLETE, ELAPSED TIME = 00:00:02
DSNU428I    DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DBHR5701.TPHR5701
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 86. Sample output of REORG TABLESPACE job with DRAIN WAIT, RETRY, and RETRY_DELAY options (Part 2 of 2)

Example 14: Using a mapping table: In the example in Figure 87 on page 495, a mapping table and mapping table index are created. Then, a REORG TABLESPACE

job uses the mapping table, and finally the mapping table is dropped. Some parts of this job use the EXEC SQL utility to execute dynamic SQL statements.

The first EXEC SQL control statement contains the SQL statements that create a mapping table that is named MYMAPPING_TABLE. The second EXEC SQL control statement contains the SQL statements that create mapping index MYMAPPING_INDEX on the table MYMAPPING_TABLE. For more information about the CREATE TABLE and CREATE INDEX statements, see *DB2 SQL Reference*.

The REORG TABLESPACE control statement then specifies that the REORG TABLESPACE utility is to reorganize table space DSN8D81P.DSN8S81C and to use mapping table MYMAPPING_TABLE.

Finally, the third EXEC SQL statement contains the SQL statements that drop MYMAPPING_TABLE. For more information about the DROP TABLE statement, see *DB2 SQL Reference*.

```
EXEC SQL
  CREATE TABLE MYMAPPING_TABLE
    (TYPE         CHAR( 01 ) NOT NULL,
     SOURCE_RID   CHAR( 05 ) NOT NULL,
     TARGET_XRID  CHAR( 09 ) NOT NULL,
     LRSN         CHAR( 06 ) NOT NULL)
   IN DSN8D81P.DSN8S81Q
   CCSID EBCDIC
ENDEXEC
EXEC SQL
  CREATE UNIQUE INDEX MYMAPPING_INDEX
   ON MYMAPPING_TABLE
    (SOURCE_RID ASC,
     TYPE,
     TARGET_XRID,
     LRSN)
   USING STOGROUP DSN8G710
   PRIQTY 120 SECQTY 20
   ERASE NO
   BUFFERPOOL BP0
   CLOSE NO
ENDEXEC
REORG TABLESPACE DSN8D81P.DSN8S81C
 COPYDDN(COPYDDN)
 SHRLEVEL CHANGE
 DEADLINE CURRENT_TIMESTAMP+8 HOURS
 MAPPINGTABLE MYMAPPING_TABLE
 MAXRO 240 LONGLOG DRAIN DELAY 900
 SORTDEVT SYSDA SORTNUM 4
 STATISTICS TABLE(ALL)
            INDEX(ALL)
EXEC SQL
  DROP TABLE MYMAPPING_TABLE
ENDEXEC
```

Figure 87. Example of creating and using a mapping table.

Example 15: Discarding records from one table while reorganizing a table space: The control statement in Figure 88 on page 496 specifies that REORG TABLESPACE is to reorganize table space DSN8D51A.DSN8S51E. During reorganization, records in table DSN8510.EMP are discarded if they have the value D11 in the WORKDEPT field. This discard criteria is specified in the WHEN clause that

follows the DISCARD option. Because a SYSDISC DD statement is included in the JCL, any discarded rows are to be written to the data set that is identified by this DD statement.

The COPYDDN option specifies that during the REORG, DB2 is also to take an inline copy of the table space. This image copy is to be written to the data set that is identified by the SYSCOPY DD statement.

```
//REORGDIS EXEC DSNUPROC,TIME=1440,
 //         UTPROC='',
 //         SYSTEM='DSN',UID='REORGDIS.EMP'
 //SYSREC   DD DISP=(NEW,CATLG,CATLG),
 //         DSN=SYSADM.REORGDIS.SYSREC,
 //         UNIT=SYSDA,SPACE=(TRK,(15,15))
 //SYSDISC  DD DISP=(NEW,CATLG,CATLG),
 //         DSN=SYSADM.REORGDIS.SYSDISC,
 //         UNIT=SYSDA,SPACE=(TRK,(15,15))
 //SYSPUNCH DD DISP=(NEW,CATLG,CATLG),
 //         DSN=SYSADM.REORGDIS.SYSPUNCH,
 //         UNIT=SYSDA,SPACE=(TRK,(15,15))
 //SYSCOPY  DD DISP=(NEW,CATLG,CATLG),
 //         UNIT=SYSDA,SPACE=(TRK,(30,30)),
 //         DSN=SYSADM.DSN8D51A.DSN8S51E.COPY
 //SYSIN    DD *
     REORG TABLESPACE
      DSN8D81A.DSN8S81E
      DISCARD
      FROM TABLE DSN8810.EMP
      WHEN (WORKDEPT = 'D11')
      SHRLEVEL NONE COPYDDN SYSCOPY
```

Figure 88. Example REORG statement that specifies discard criteria

Example 16: Discarding records from multiple tables while reorganizing a table space: The control statement in Figure 89 on page 497 specifies that REORG TABLESPACE is to reorganize table space DBKC0501.TLKC0501. During reorganization, the following records are discarded:

- Records in table TBKC0501 that have a value in the QT_INV_TRANSACTION column that is less than or equal to 700, and a value in the NO_DEPT column that is equal to X'33303230'.
- Records in table TBKC0502 that have a value in the NO_WORK_CENTER column that is equal to either X'333031303120' or X'333032303620'.

This discard criteria is specified with the DISCARD option. Any discarded rows are to be written to the SYSDISC data set, as specified by the DISCARDDN option.

```
//STEP1    EXEC DSNUPROC,UID='IUKCU105.REORG2',
//         UTPROC='',
//         SYSTEM='SSTR'
//UTPRINT  DD  SYSOUT=*
//SYSDISC  DD DSN=IUKCU105.REORG2.STEP1.SYSDISC,
//         DISP=(MOD,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(2000,(20,20),,,ROUND),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400)
//SYSREC   DD DSN=IUKCU105.REORG2.STEP1.SYSREC,
//         DISP=(MOD,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSCOPY  DD DSN=IUKCU105.REORG2.STEP1.SYSCOPY,
//         DISP=(MOD,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//LOADSTMT DD DSN=IUKCU105.REORG2.STEP1.SYSPUNCH,
//         DISP=(MOD,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
    REORG TABLESPACE DBKC0501.TLKC0501 SHRLEVEL REFERENCE
          PUNCHDDN LOADSTMT DISCARDDN SYSDISC
          UNLOAD CONTINUE
          DISCARD
          FROM TABLE TBKC0501
             WHEN (QT_INV_TRANSACTION <= 700 AND
                   NO_DEPT = X'33303230')
          FROM TABLE TBKC0502
           WHEN (NO_WORK_CENTER = X'333031303120' OR
                 NO_WORK_CENTER = X'333032303620')
/*
```

Figure 89. Example REORG statement that specifies discard criteria for several tables

Example 17: Reorganizing only those partitions that are in REORG-pending status. The control statement in Figure 90 specifies that REORG TABLESPACE is to reorganize only those partitions of table space DBKQAA01.TPKQAA01 that are in the range from 2 to 10 and are in REORG-pending status.

```
//STEP1    EXEC DSNUPROC,UID='JUKQU1AA.REORG6',
//            UTPROC='',SYSTEM='SSTR'
//SYSREC   DD DSN=JUKQU1AA.REORG6.STEP1.SYSREC,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSCOPY  DD DSN=JUKQU1AA.REORG6.STEP1.SYSCOPY,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSUT1   DD DSN=JUKQU1AA.REORG6.STEP1.SYSUT1,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT  DD DSN=JUKQU1AA.REORG6.STEP1.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
    REORG TABLESPACE DBKQAA01.TPKQAA01 SCOPE PENDING PART 2:10
/*
```

Figure 90. Example REORG TABLESPACE statement with SCOPE PENDING

Chapter 26. REPAIR

The online REPAIR utility repairs data. The data can be your own data or data that you would not normally access, such as space map pages and index entries.

You use REPAIR to replace invalid data with valid data. Be extremely careful when using REPAIR. Improper use can damage the data even further.

You can use the REPAIR utility to:
- Test database definitions (DBDs)
- Repair DBDs
- Reset a pending status on a table space or index
- Verify the contents of data areas in table spaces and indexes
- Replace the contents of data areas in table spaces and indexes
- Delete a single row from a table space
- Produce a hexadecimal dump of an area in a table space or index
- Delete an entire LOB from a LOB table space
- Dump LOB pages
- Rebuild object descriptors (OBDs) for a LOB table space
- Manage version numbers

For a diagram of REPAIR syntax and a description of available options, see "Syntax and options of the REPAIR control statement" on page 500. For detailed guidance on running this utility, see "Instructions for running REPAIR" on page 514.

Output: The output from the REPAIR utility can consist of one or more modified pages in the specified DB2 table space or index and a dump of the contents.

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:
- REPAIR privilege for the database
- DBADM or DBCTRL authority for the database
- SYSCTRL or SYSADM authority

An ID with installation SYSOPR authority can also execute REPAIR, but only on a table space in the DSNDB01 or DSNDB06 database.

To execute REPAIR with the DBD option, you must use a privilege set that includes SYSADM, SYSCTRL, or installation SYSOPR authority.

REPAIR should be used only by a person that is knowledgeable in DB2 and your data. Grant REPAIR authorization only to the appropriate people.

Execution phases of REPAIR: The phases for REPAIR are:

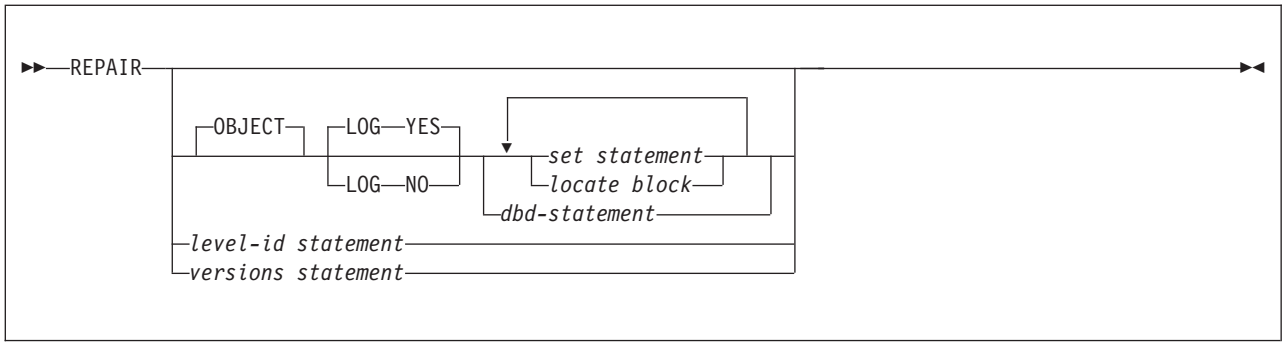
Phase	Description
UTILINIT	Performs initialization
REPAIR	Repairs data
UTILTERM	Performs cleanup

The following topics provide additional information:

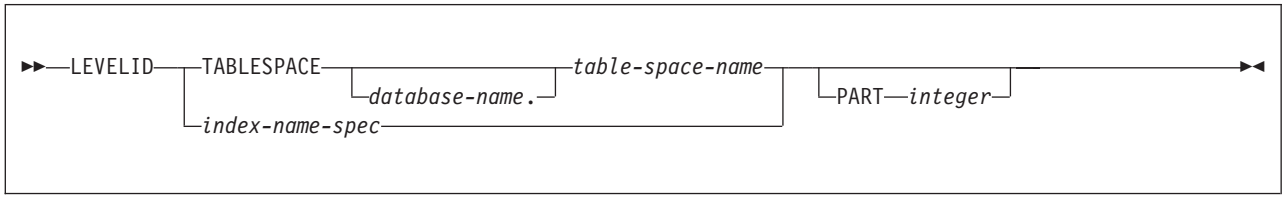
Syntax and options of the REPAIR control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

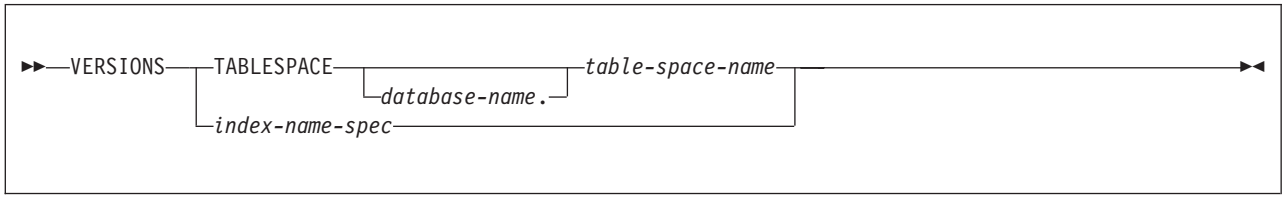
REPAIR syntax diagram



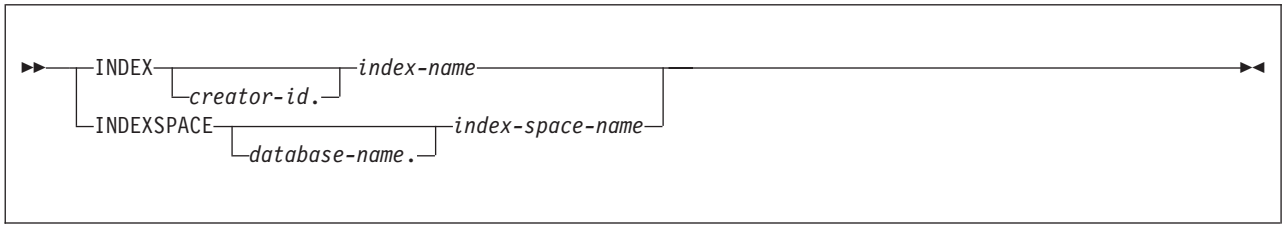
level-id statement:



versions statement:



index-name-spec:



REPAIR option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

OBJECT
Indicates that an object is to be repaired. This keyword Is optional. only.

LOG
Indicates whether the changes that REPAIR makes are to be logged. If the changes are to be logged, they are applied again if the data is recovered.

YES
Indicates that the changes are to be logged. The **default** is **YES**.

REPAIR LOG YES cannot override the LOG NO attribute of a table space.

NO
Indicates that the changes are not to be logged. You cannot use this option with a DELETE statement.

REPAIR LOG NO can override the LOG YES attribute of a table space.

LEVELID
Indicates that the level identifier of the named table space, table space partition, index, or index space partition is to be reset to a new identifier. Use LEVELID to accept the use of a down-level data set. You cannot specify multiple LEVELID keywords in the same REPAIR control statement.

You cannot use LEVELID with a table space, table space partition, index, or index space partition that has outstanding indoubt log records or pages in the logical page list (LPL).

Attention: Accepting the use of a down-level data set might cause data inconsistencies. Problems with inconsistent data that result from resetting the level identifier are the responsibility of the user.

TABLESPACE *database-name.table-space-name*
Specifies the table space (and, optionally, the database to which it belongs) whose level identifier is to be reset (if you specify LEVELID) or whose version identifier is to be updated (if you specify VERSIONS).

database-name
Specifies the name of the database to which the table space belongs. The **default** is **DSNDB04**.

table-space-name
Specifies the name of the table space.

INDEX
Specifies the index whose level identifier is to be reset (if you specify LEVELID) or whose version identifier is to be updated (if you specify VERSIONS).

creator-id.
Specifies the creator of the index. Specifying this qualifier is optional.

index-name
Specifies the name of the index. Enclose the index name in quotation marks if the name contains a blank.

You can specify either INDEX or INDEXSPACE to identify an index. To specify multiple indexes, repeat the keyword.

INDEXSPACE
Specifies the index space for the index whose level identifier is to be reset (if you specify LEVELID) or whose version identifier is to be updated (if you specify VERSIONS). You can obtain the index space name for an index from the SYSIBM.SYSINDEXES catalog table. The index space name must be qualified.

database-name.
Specifies the name of the database to which the index space belongs.

index-space-name
Specifies the name of the index space.

You can specify either INDEX or INDEXSPACE to identify an index. To specify multiple indexes, repeat the keyword.

PART
Identifies a partition of the table space or index (including a partition of a data-partitioned secondary index).

integer is the number of the partition and must be in the range from one to the number of partitions that are defined for the object. The maximum is 4096.

VERSIONS
Updates the version information in the catalog and directory for the specified table space or index with the version information from the system pages of the object. Use REPAIR VERSIONS in the following situations:

- When you run the DSN1COPY utility with the OBIDXLAT option to move objects from one system to another. For more information about this process, see "Updating version information when moving objects to another subsystem" on page 518.
- Run REPAIR VERSIONS only when moving objects.

For more information about version number management, see Part 2 of *DB2 Administration Guide*.

SET statement syntax

The SET TABLESPACE statement resets the COPY-pending, RECOVER-pending, CHECK-pending, auxiliary warning (AUXW), and auxiliary CHECK-pending (ACHKP) statuses for a table space or data set. The SET INDEX statement resets the informational COPY-pending (ICOPY), RECOVER-pending, REBUILD-pending, or CHECK-pending status for an index.

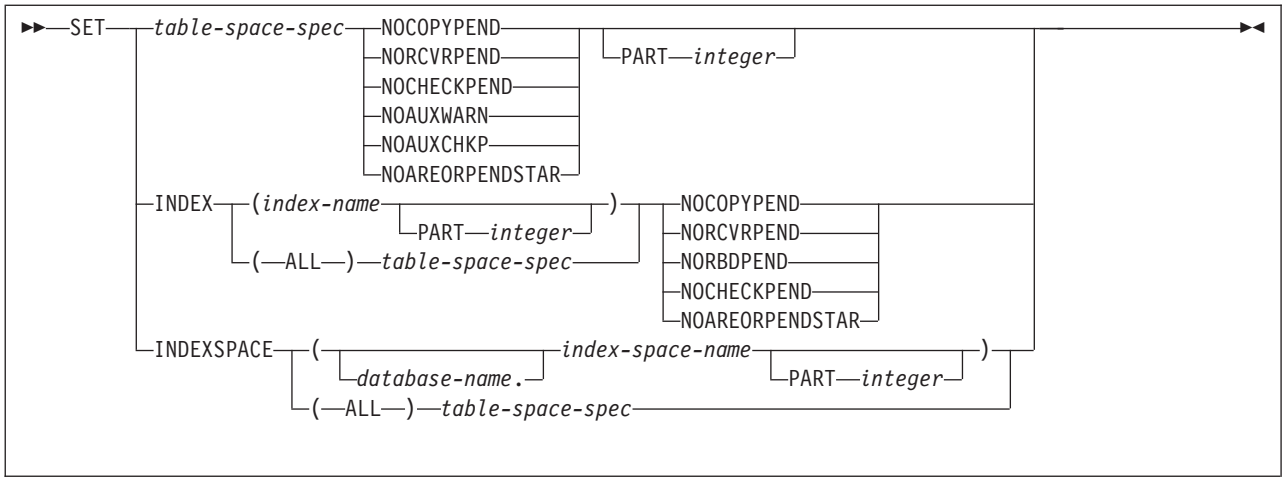
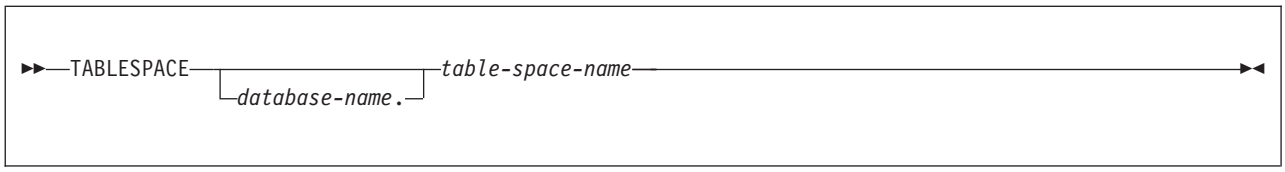


table-space-spec:



SET statement option descriptions

SET TABLESPACE *database-name.table-space-name*
Specifies the table space (and, optionally, the database to which it belongs) whose pending status is to be reset.

database-name
Specifies the name of the database to which the table space belongs. The **default** is **DSNDB04**.

table-space-name
Specifies the name of the table space.

SET INDEX
Specifies the index whose RECOVER-pending, CHECK-pending, REBUILD-pending, or informational COPY-pending status is to be reset.

(*index-name***)**
Specifies the index that is to be processed. Enclose the index name in quotation marks if the name contains a blank.

(ALL) Specifies that all indexes in the table space will be processed.

The keyword INDEXES is accepted following a *table-space-spec* and causes all indexes to be processed. You can also repair all indexes by specifying INDEX(ALL) followed by a *table-space-spec*.

SET INDEXSPACE
Specifies the index space for the index whose RECOVER-pending, CHECK-pending, REBUILD-pending, or informational COPY-pending status is to be reset.

(*database-name.index-space-name***)**
Specifies the index space that is to be processed.

(ALL) Specifies that all indexes in the table space will be processed.

PART *integer*
Specifies a particular partition whose COPY-pending, informational COPY-pending, or RECOVER-pending status is to be reset. If you do not specify PART, REPAIR resets the pending status of the entire table space or index.

integer is the number of the partition and must be in the range from one to the number of partitions that are defined for the object. The maximum is 4096.

You can specify PART for NOCHECKPEND on a table space, and for NORCVRPEND on indexes.

The PART keyword is not valid for a LOB table space or an index on the auxiliary table.

NOCOPYPEND
Specifies that the COPY-pending status of the specified table space, or the informational COPY-pending (ICOPY) status of the specified index is to be reset.

NORCVRPEND
Specifies that the RECOVER-pending (RECP) status of the specified table space or index is to be reset.

NORBDPEND
Specifies that the REBUILD-pending (RBDP) status, the page set REBUILD-pending status (PSRBDP), or the RBDP* status of the specified index is to be reset.

NOCHECKPEND
Specifies that the CHECK-pending (CHKP) status of the specified table space or index is to be reset.

NOAUXWARN
Specifies that the auxiliary warning (AUXW) status of the specified table space is to be reset. The specified table space must be a base table space or a LOB table space.

NOAUXCHKP
Specifies that the auxiliary CHECK-pending (ACHKP) status of the specified table space is to be reset. The specified table space must be a base table space.

NOAREORPENDSTAR
Resets the advisory REORG-pending (AREO*) status of the specified table space or index.

LOCATE block syntax

A LOCATE block is a set of statements, each with its own options, that begins with a LOCATE statement and ends with the next LOCATE or SET statement, or with the end of the job. You can include more than one LOCATE block in a REPAIR utility statement.

In any LOCATE block, you can use VERIFY, REPLACE, or DUMP as often as you like; you can use DELETE only once.

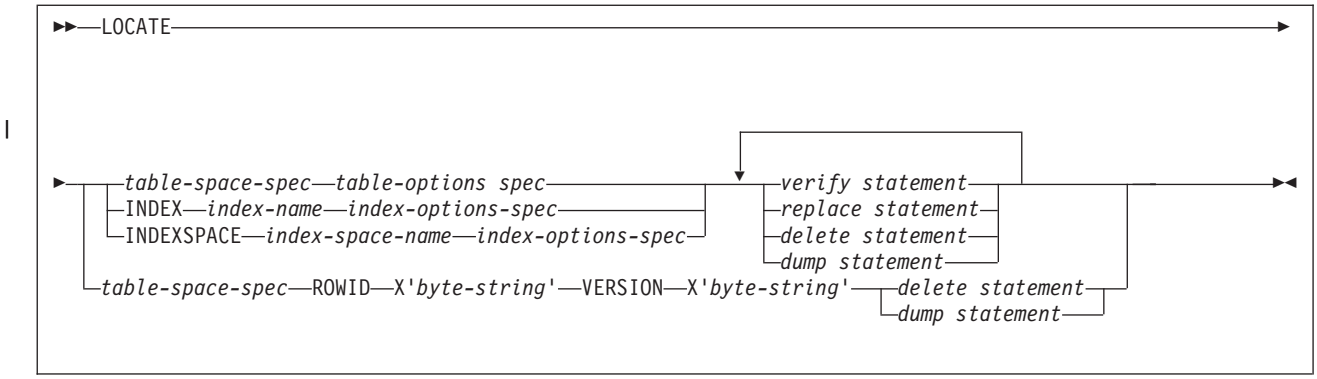


table-space-spec:

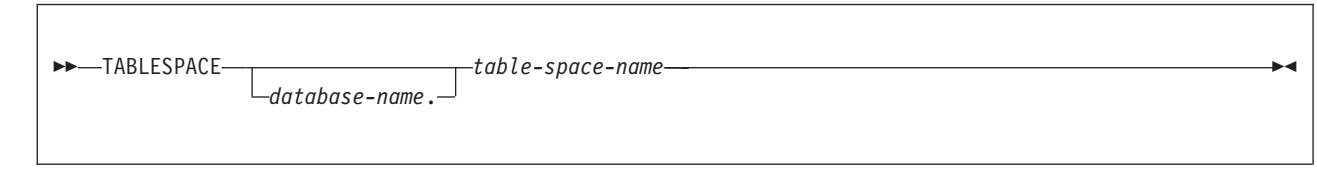
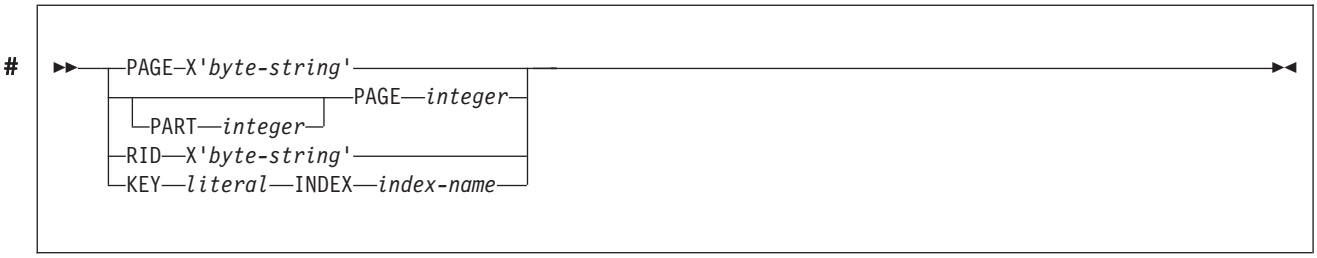
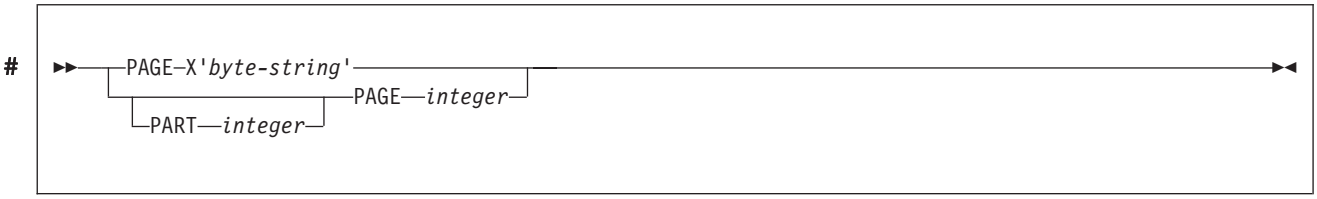


table-options-spec:



index-options-spec:



LOCATE TABLESPACE statement option descriptions

The LOCATE TABLESPACE statement locates data that is to be repaired within a table space.

One LOCATE statement is required for each unit of data that is to be repaired. Several LOCATE statements can appear after each REPAIR statement.

If a REPAIR statement is followed by more than one LOCATE statement, all processing that is caused by VERIFY, REPLACE, and DUMP statements is committed before the next LOCATE statement is processed.

TABLESPACE *database-name.table-space-name*
Specifies the table space (and, optionally, the database to which it belongs) in which data is to be located for repair.

database-name
Is the name of the database to which the table space belongs and is optional. The **default** is **DSNDB04**.

table-space-name
Is the name of the table space that contains the data that you want to repair.

PART *integer*
Specifies the partition that contains the page that is to be located. Part is valid only for partitioned table spaces.

integer is the number of the partition.

PAGE
Specifies the relative page number within the table space, partitioned table space, or index that is to be operated on. The first page, in either case, is 0 (zero).

integer *integer* is a decimal number from one to six digits in length.

X'*byte-string***'**
Specifies that the data of interest is an entire page. The specified offsets in *byte-string* and in subsequent statements are relative to the beginning of the page. The first byte of the page is at offset 0.

byte-string is a hexadecimal value from one to eight characters in length. You do not need to enter leading zeros. Enclose the *byte-string* between apostrophes, and precede it with X.

RID X'*byte-string***'**
Specifies that the data that is to be located is a single row. The specified offsets in *byte-string* and in subsequent statements are relative to the beginning of the row. The first byte of the stored row prefix is at offset 0.

byte-string can be a hexadecimal value from one to eight characters in length. You do not need to enter leading zeros. Enclose the byte string between apostrophes, and precede it with an X.

KEY *literal*
Specifies that the data that is to be located is a single row, identified by *literal*. The specified offsets in subsequent statements are relative to the beginning of the row. The first byte of the stored row prefix is at offset 0.

literal is any SQL constant that can be compared with the key values of the named index.

Character constants that are specified within the LOCATE KEY option cannot be specified as ASCII or Unicode character strings. No conversion of the values is performed. To use this option when the table space is ASCII or Unicode, you should specify the values as hexadecimal constants.

If more than one row has the value *literal* in the key column, REPAIR returns a list of record identifiers (RIDs) for records with that key value, but does **not** perform any other operations (verify, replace, delete, or dump) until the next LOCATE TABLESPACE statement is encountered. To repair the proper data, write a LOCATE TABLESPACE statement that selects the desired row, using the RID option, the PAGE option, or a different KEY and INDEX option. Then execute REPAIR again.

ROWID X'*byte-string***'**
Specifies that the data that is to be located is a LOB in a LOB table space.

byte-string is the row ID that identifies the LOB column.

Use the ROWID keyword to repair an orphaned LOB row. You can find the ROWID in the output from the CHECK LOB utility. If you specify the ROWID keyword, the specified table space must be a LOB table space.

VERSION X'*byte-string***'**
Specifies that the data that is to be located is a LOB in a LOB table space.

byte-string is the version number that identifies the version of the LOB column.

Use the VERSION keyword to repair an orphaned LOB column. You can find the VERSION number in the output of the CHECK LOB utility or an out-of-synch LOB that is reported by the CHECK DATA utility. If you specify the VERSION keyword, the specified table space must be a LOB table space.

LOCATE INDEX statement and LOCATE INDEXSPACE statement option descriptions

The LOCATE INDEX (or INDEXSPACE) statement locates data that is to be repaired within an index. You can specify indexes by either their index name or their index space name.

One LOCATE statement is required for each unit of data that is to be repaired. Multiple LOCATE statements can appear after each REPAIR statement.

If a REPAIR statement is followed by multiple LOCATE statements, all processing that is caused by VERIFY, REPLACE, and DUMP statements is committed before the next LOCATE statement is processed.

INDEX *index-name*
Specifies a particular index that is to be used to find the row that contains the key. When you are locating an index by key, the index that you specify must be a single-column index.

index-name is the qualified or unqualified name of the index. If you omit the qualifier creator ID, the user identifier for the utility job is used. Enclose the index name in quotation marks if the name contains a blank.

INDEXSPACE *index-space-name*
Specifies the index space for a particular index that is to be used to find the row that contains the key. Look in the SYSIBM.SYSINDEXES catalog table to find the index space name for an index. When you are locating an index by key, the index that you specify must be a single-column index.

index-space-name is the qualified name of the index space, in the form *database-name.index-space-name*.

PART *integer*
Specifies the partition number of the partitioning index that contains the page that is to be located. The PART keyword is valid only for indexes of partitioned table spaces.

integer is the number of the partitioning index.

PAGE *integer*
Specifies the relative page number within the index space that is to be operated on. The first page is 0 (zero).

integer *integer* is a decimal number from one to six digits in length.

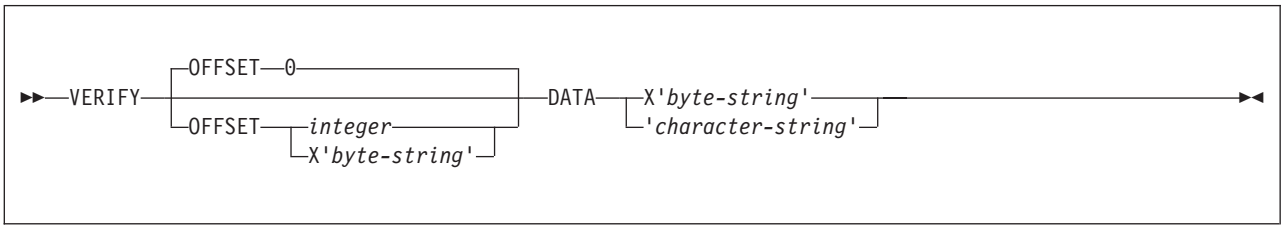
X'*byte-string***'**
Specifies that the data of interest is an entire page. The specified offsets in *byte-string* and in subsequent statements are relative to the beginning of the page. The first byte of the page is at offset 0.

byte-string is a hexadecimal value from one to eight characters in length. You do not need to enter leading zeros. Enclose the *byte-string* between apostrophes, and precede it with X.

VERIFY statement syntax

The VERIFY statement tests whether a particular data area contains a specified value. Depending on the outcome of this test, the REPAIR utility performs the following actions:

- If the data area does contain the value, subsequent operations in the same LOCATE block are allowed to proceed.
- If **any** data area does not contain its specified value, **all** subsequent operations in the same LOCATE block are inhibited.



VERIFY statement option descriptions

OFFSET
Locates the data that is to be tested by a relative byte address (RBA) within the row or page.

integer Identifies the offset as an integer. The **default** is **0**, the first byte of the area that is identified by the previous LOCATE statement.

X'*byte-string***'**
Identifies the offset as one to four hexadecimal characters. You do not need to enter leading zeros. Enclose the byte string between apostrophes, and precede it with X.

DATA
Specifies what data must be present at the current location before a change is made.

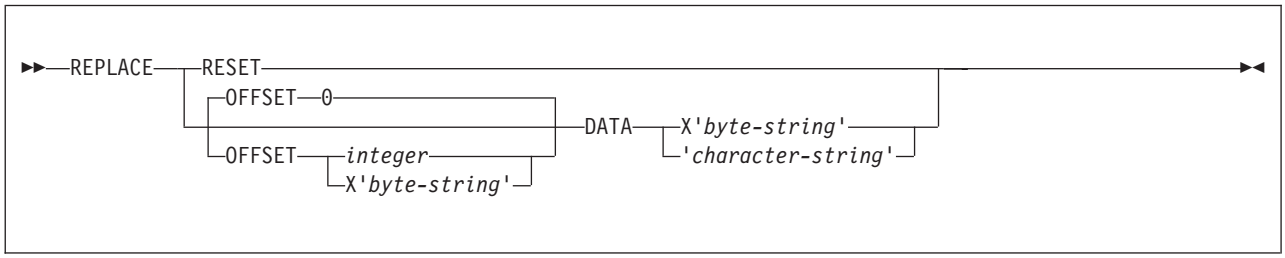
Character constants that are specified within the VERIFY DATA option cannot be specified as ASCII or Unicode character strings. No conversion of the values is performed. To use this option when the table space is ASCII or Unicode, you should specify the values as hexadecimal constants.

X'*byte-string***'**
Specifies an even number of hexadecimal characters that must be present. You do not need to enter leading zeros. Enclose the byte string between apostrophes, and precede it with X.

'*character-string***'**
Specifies any character string that must be present.

REPLACE statement syntax

The REPLACE statement replaces data at a particular location. The statement is contained within a LOCATE block. If any VERIFY statement within that block finds a data area that does not contain its specified data, the REPLACE operation is inhibited.



REPLACE statement option descriptions

RESET
Specifies that the inconsistent data indicator is to be reset. A page for which this indicator is on is considered in error, and the indicator must be reset before you can access the page. Numbers of pages with inconsistent data are reported at the time that they are encountered.

The option also resets the PGCOMB flag bit in the first byte of the page to agree with the bit code in the last byte of the page.

OFFSET
Indicates where data is to be replaced by a relative byte address (RBA) within the row or page. Only one OFFSET and one DATA specification are acted on for each REPLACE statement.

integer Specifies the offset as an integer. The **default** is **0**, the first byte of the area that is identified by the previous LOCATE statement.

X'*byte-string***'**
Specifies the offset as one to four hexadecimal characters. You do not need to enter leading zeros. Enclose the byte string between apostrophes, and precede it with X.

DATA
Specifies the new data that is to be entered. Only one OFFSET and one DATA specification are acted on for each REPLACE statement.

Character constants that are specified within the VERIFY DATA option cannot be specified as ASCII or Unicode character strings. No conversion of the values is performed. To use this option when the table space is ASCII or Unicode, you should specify the values as hexadecimal constants.

X'*byte-string***'**
Specifies an even number of hexadecimal characters that are to replace the current data. You do not need to enter leading zeros. Enclose the byte string between apostrophes, and precede it with X.

'*character-string***'**
Specifies any character string that is to replace the current data.

DELETE statement syntax and description

The DELETE statement deletes a single row of data that has been located by a RID or KEY option. The statement is contained within a LOCATE block. If any VERIFY statement within that block finds a data area that does not contain its specified data, the DELETE operation is inhibited.

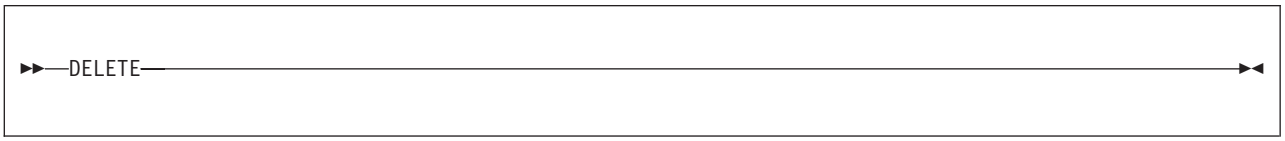
The DELETE statement operates without regard for referential constraints. If you delete a parent row, its dependent rows remain unchanged in the table space. However, in the DB2 catalog and directory table spaces, where links are used to reference other tables in the catalog, deleting a parent row causes all child rows to be deleted, as well. Moreover, deleting a child row in the DB2 catalog tables also updates its predecessor and successor pointer to reflect the deletion of this row. Therefore, if the child row has incorrect pointers, the DELETE might lead to an unexpected result. See "Example 5: Repairing a table space with an orphan row" on page 523 for a possible method of deleting a child row without updating its predecessor and successor pointer.

In any LOCATE block, you can include no more than one DELETE option.

If you have coded any of the following options, you cannot use DELETE:

- The LOG NO option on the REPAIR statement
- A LOCATE INDEX statement to begin the LOCATE block
- The PAGE option on the LOCATE TABLESPACE statement in the same LOCATE block
- A REPLACE statement for the same row of data

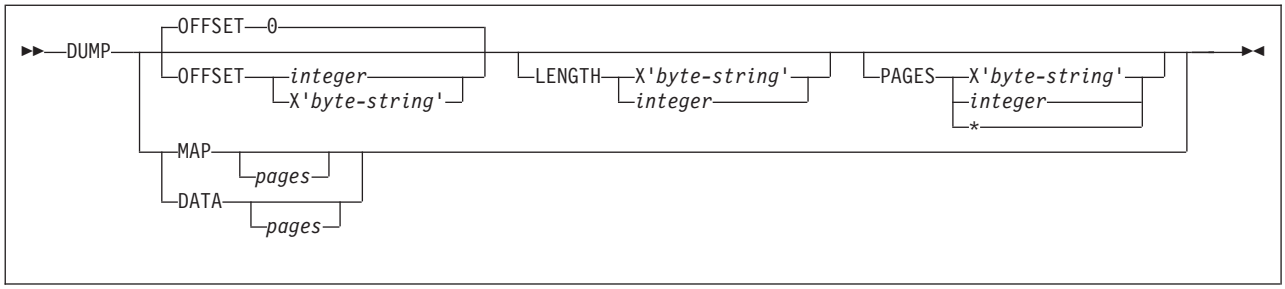
When you specify LOCATE ROWID for a LOB table space, the LOB that is specified by ROWID is deleted with its index entry. All pages that are occupied by the LOB are converted to free space. The DELETE statement **does not** remove any reference to the deleted LOB from the base table space.



DUMP statement syntax

The DUMP statement produces a hexadecimal dump of data that is identified by offset and length. DUMP statements have no effect on VERIFY or REPLACE operations.

When you specify LOCATE ROWID for a LOB table space, one or more map or data pages of the LOB are dumped. The DUMP statement dumps all of the LOB column pages if you do not specify either the MAP or DATA keyword.



DUMP statement option descriptions

OFFSET
Optionally, locates the data that is to be dumped by a relative byte address (RBA) within the row or page.

integer Specifies the offset as an integer. The **default** is **0**, the first byte of the row or page.

X'*byte-string***'**
Specifies the offset as one to four hexadecimal characters. You do not need to enter leading zeros. Enclose the byte string between apostrophes, and precede it with X.

LENGTH
Optionally, specifies the number of bytes of data that are to be dumped. If you omit both LENGTH and PAGE, the dump begins at the specified OFFSET and continues to the end of the row or page.

If you specify a number of bytes (with LENGTH) and a number of pages (with PAGE), the dump contains the same relative bytes from each page. That is, from each page you see the same number of bytes, beginning at the same offset.

X'*byte-string***'**
Specifies one to four hexadecimal characters. You do not need to enter leading zeros. Enclose the byte string between apostrophes, and precede it with X.

integer Specifies the length as an integer.

PAGES
Optionally, specifies a number of pages that are to be dumped. You can use this option only if you used PAGE in the preceding LOCATE TABLESPACE control statement.

X'*byte-string***'**
Specifies one to four hexadecimal characters. You do not need to enter leading zeros. Enclose the byte string between apostrophes, and precede it with X.

integer Specifies the number of pages as an integer.

***** Specifies that all pages from the starting point to the end of the table space or partition are to be dumped.

MAP *pages*
Specifies that only the LOB map pages are to be dumped.

pages specifies the number of LOB map pages that are to be dumped. If you do not specify *pages*, all LOB map pages of the LOB that is specified by ROWID and version are dumped.

DATA *pages*
Specifies that only the LOB data pages are to be dumped.

pages specifies the number of LOB data pages that are to be dumped. If you do not specify *pages*, all LOB data pages of the LOB that is specified by ROWID and version are dumped.

DBD statement syntax

When REPAIR DBD REBUILD is running, an S-lock is acquired for the appropriate
catalog tables. If the S-lock fails, REPAIR DBD fails.

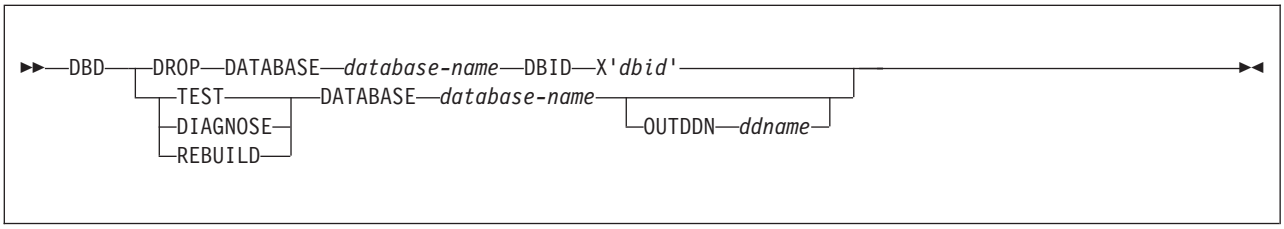
The DBD statement allows you to:

- Compare the database definition (DBD) in the DB2 catalog with its definition in the DB2 directory
- Rebuild a database definition in the directory by using the information including LOB information in the DB2 catalog
- Drop an inconsistent database definition from the DB2 catalog and the DB2 directory

The REPAIR utility assumes that the links in table spaces DSNDB01.DBD01, DSNDB06.SYSDBAUT, and DSNDB06.SYSDBASE are intact. Before executing REPAIR with the DBD statement, run the DSN1CHKR utility on these table spaces to ensure that the links are not broken. For more information about DSN1CHKR, see Chapter 38, "DSN1CHKR," on page 709.

The database on which REPAIR DBD REBUILD is run must be started for access by utilities only. DB2 performs this step automatically. For more information about using the DBD statement, see "Using the DBD statement" on page 516.

You can use REPAIR DBD on declared temporary tables, which must be created in a database that is defined with the AS TEMP clause. No other DB2 utilities can be used on a declared temporary table, its indexes, or its table spaces.



DBD statement option descriptions

DROP
Specifies that the named database is to be dropped from both the DB2 catalog and the DB2 directory. When you specify this option, DB2 also drops databases that contain tables that have been created with RESTRICT ON DROP. Use this keyword if the SQL DROP DATABASE statement fails because the description of the database is not in both the DB2 catalog and the DB2 directory. If you cannot use the ALTER command to remove the with RESTRICT ON DROP

option on tables in a database that is badly damaged and you need to drop the database, you can use this keyword to drop the database.

Attention: Use the DROP option with extreme care. Using DROP can cause additional damage to your data. For more assistance, you can contact IBM Software Support.

DATABASE *database-name*
Specifies the target database.

database-name is the name of the target database, which cannot be DSNDB01 (the DB2 directory) or DSNDB06 (the DB2 catalog).

If you use TEST, DIAGNOSE, or REBUILD, *database-name* cannot be DSNDB07 (the work file database).

If you use DROP, *database-name* cannot be DSNDB04 (the default database).

DBID X'*dbid***'**
Specifies the database descriptor identifier for the target database.

dbid is the database descriptor identifier.

TEST
Specifies that a DBD is to be built from information in the DB2 catalog, and is to be compared with the DBD in the DB2 directory. If you specify TEST, DB2 reports significant differences between the two DBDs.

If the condition code is 0, the DBD in the DB2 directory is consistent with the information in the DB2 catalog.

If the condition code is not 0, then the information in the DB2 catalog and the DBD in the DB2 directory might be inconsistent. Run REPAIR DBD with the DIAGNOSE option to gather information that is necessary for resolving any possible inconsistency.

DIAGNOSE
Specifies that information that is necessary for resolving an inconsistent database definition is to be generated. Like the TEST option, DIAGNOSE builds a DBD that is based on the information in the DB2 catalog and compares it with the DBD in the DB2 directory. In addition, DB2 reports any differences between the two DBDs, and produces hexadecimal dumps of the inconsistent DBDs.

If the condition code is 0, the information in the DB2 catalog and the DBD in the DB2 directory is consistent.

If the condition code is 8, the information in the DB2 catalog and the DBD in the DB2 directory might be inconsistent.

For further assistance in resolving any inconsistencies, you can contact IBM Software Support.

REBUILD
Specifies that the DBD that is associated with the specified database is to be rebuilt from the information in the DB2 catalog.

Attention: Use the REBUILD option with extreme care, as you can cause more damage to your data. For more assistance, you can contact IBM Software Support.

OUTDDN *ddname*
Specifies the DD statement for an optional output data set. This data set contains copies of the DB2 catalog records that are used to rebuild the DBD.

ddname is the name of the DD statement.

Instructions for running REPAIR

To run REPAIR, you must:

1. Read "Before running REPAIR" in this section.
2. Prepare the necessary data sets, as described in "Data sets that REPAIR uses."
3. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for REPAIR, see "Sample REPAIR control statements" on page 522.)
4. Prepare a utility control statement that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks" on page 515.
5. Check the compatibility table in "Concurrency and compatibility for REPAIR" on page 519 if you want to run other jobs concurrently on the same target objects.
6. Run REPAIR by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Attention: Be extremely careful when using the REPAIR utility to replace data. Changing data to invalid values by using REPLACE might produce unpredictable results, particularly when changing page header information. Improper use of REPAIR can result in damaged data, or in some cases, system failure.

Before running REPAIR

Before you run the REPAIR utility, perform the following actions.

Making a copy of the table space

Before starting to use REPAIR to change data, ensure that you have a copy (full image copy or DSN1COPY generated copy) of the affected table space to enable fallback.

Restoring damaged indexes

Because REPAIR can access index data only by referring to a page and an offset within the page, identifying and correcting a problem can be difficult. Use REBUILD INDEX or RECOVER INDEX to restore damaged index data.

Running REPAIR on encrypted data

Do not run REPAIR on encrypted data. REPAIR does not decrypt the data. The utility reads the data in its encrypted form and then manipulates the data without decrypting it.

Data sets that REPAIR uses

Table 92 lists the data sets that REPAIR uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 92. Data sets that REPAIR uses

Data set	Data set	Required?
SYSIN	Input data set that contains the utility control statement.	Yes

Table 92. Data sets that REPAIR uses (continued)

Data set	Data set	Required?
SYSPRINT	Output data set for messages.	Yes
Optional output data set	Data set that contains copies of the DB2 catalog records that are used to rebuild the DBD. You define the DD name.	No

The following objects are named in the utility control statement and do not require a DD statement in the JCL:

Table space or index
Object that is to be repaired.

Calculating output data set size: Use the following formula to estimate the size of the output data set:

```
SPACE = (4096,(n,n))
```

In this formula, *n* = the total number of DB2 catalog records that relate to the database on which REPAIR DBD is being executed.

You can calculate an estimate for *n* by summing the results of SELECT COUNT(*) from all of the catalog tables in the SYSDBASE table space, where the name of the database that is associated with the record matches the database on which REPAIR DBD is being executed.

Creating the control statement

Create the utility control statement for the REPAIR job. See "Syntax and options of the REPAIR control statement" on page 500 for REPAIR syntax and option descriptions. See "Sample REPAIR control statements" on page 522 for examples of REPAIR usage.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in your utility control statement:

Resetting table space status

In most cases, resetting the COPY-pending restriction by taking a full image copy is preferable to using REPAIR. This is because RECOVER cannot be executed successfully until an image copy has been made.

Resetting the RECOVER-pending status by running RECOVER or LOAD is preferable to using REPAIR. This is because RECOVER uses DB2-controlled recovery information, whereas REPAIR SET TABLESPACE or INDEX resets the

RECOVER-pending status without considering the recoverability of the table space. Recoverability issues include the availability of image copies, of rows in SYSIBM.SYSCOPY, and of log data sets.

Verifying and possibly correcting referential integrity constraints by running CHECK DATA are recommended. CHECK DATA performs a complete check of all referential integrity constraints of the table space set, whereas with REPAIR, you are responsible for checking all the referential integrity constraints violations.

To reset the CHECK-pending status for a LOB table space:

1. Run the CHECK DATA utility again with the AUXERROR INVALIDATE keywords specified.
2. Update the invalid LOBs.

To reset the auxiliary warning (AUXW) status for a LOB table space:

1. Update or correct the invalid LOB columns, then
2. Run the CHECK LOB utility with the AUXERROR INVALIDATE option if invalid LOB columns were corrected.

Resetting index space status

Running COPY INDEXSPACE to reset the informational COPY-pending status is preferable to using the REPAIR utility to reset the status.

Consider using the REBUILD INDEX or RECOVER INDEX utility on an index that is in REBUILD-pending status, rather than running REPAIR SET INDEX NORBDPEND. RECOVER uses DB2-controlled recovery information, whereas REPAIR SET INDEX resets the REBUILD-pending status without considering the recoverability of the index. Recoverability issues include the availability of image copies, of rows in SYSIBM.SYSCOPY, and of log data sets.

Repairing a damaged page

1. Execute REPAIR with the LOG YES option and the DUMP control statement, specifying the pages that you suspect are damaged. Verify that the dump you receive contains the desired pages.
2. If you know which page is damaged and you can see how to resolve the error, repair the page and reset the “inconsistent data” indicator. Run REPAIR with the REPLACE RESET DATA control statement. Document your actions in case you need to undo anything later.
3. If you determine that the page is not **really** damaged, but merely has the “inconsistent data” indicator on, reset the indicator by running REPAIR with the REPLACE RESET control statement.

Using the DBD statement

To use the DBD statement:

1. Run the DSN1CHKR utility on the DSNDB01.DBD01, DSNDB06.SYSDBAUT, and DSNDB06.SYSDBASE table spaces.
2. Run REPAIR DBD with the TEST option to determine if the information in the DB2 catalog is consistent with the DBD in the DB2 directory. REPAIR DBD TEST obtains environment information, such as the character that is used for the decimal point, from the DSNHDECP module for the subsystem.
3. If inconsistencies exist (condition code is not 0), use the DIAGNOSE option with the OUTDDN keyword to produce diagnostic information. Contact IBM Software Support for assistance in analyzing this information. REPAIR DBD

DIAGNOSE obtains environment information, such as the character that is used for the decimal point, from the DSNHDECP module for the subsystem.

4. If IBM Software Support instructs you to do so, replace the existing DBD with the REBUILD option. **Do not** use this option if you suspect that information in the catalog is causing the inconsistency. REBUILD uses information in the catalog to rebuild the DBD; if the catalog is incorrect, the rebuilt DBD cannot be correct.

   DB2 reads each table space in the database during the REBUILD process to gather information. If the data sets for the table spaces do not exist or are not accessible to DB2, the REBUILD abnormally terminates.

   REPAIR DBD REBUILD obtains environment information, such as the character that is used for the decimal point, from the DSNHDECP module for the subsystem.

5. If you suspect an inconsistency in the DBD of the work file database, run REPAIR DBD DROP or DROP DATABASE (SQL), and then recreate it. If you receive errors when you drop the work file database, contact IBM Software Support for assistance.

6. If the database is started for utility-only access, issue the STOP DATABASE
(database-name) command, and then issue the START DATABASE
(database-name) ACCESS(RW) command to allow full access to the database.
You need to complete this step if you ran REPAIR DBD REBUILD. DB2 starts
the database for utility-only access when you execute this utility.

Locating rows by key

If you use LOCATE TABLESPACE KEY, a number of rows might satisfy the condition. In this case, REPAIR returns only the RIDs of the rows and does not perform any VERIFY, REPLACE, DELETE, or DUMP actions which might be coded in that LOCATE block. You can then use the RID option of LOCATE TABLESPACE to identify a specific row. Examples of the messages that are issued are shown in the following example:

```
DSNU658I - DSNUCBRL - MULTIPLE RECORDS FOUND WITH SPECIFIED KEY
DSNU660I - DSNUCBRL - POSSIBLE RID - X00000100B'
DSNU660I - DSNUCBRL - POSSIBLE RID - X000000C18'
DSNU660I - DSNUCBRL - POSSIBLE RID - X000000916'
DSNU660I - DSNUCBRL - POSSIBLE RID - X000000513'
DSNU650I - DSNUCBRP - DUMP
DSNU012I   DSNUGBAC - UTILITY EXECUTION TERMINATED,
                      HIGHEST RETURN CODE=8
```

Multiple-column indexes: The KEY option supports only single-column indexes. The following message is issued if you try to locate a row by using a multiple-column index.

```
DSNUCBRK - INDEX USED HAS MULTIPLE-FIELD KEY
```

Using VERIFY with REPLACE and DELETE operations

If **any** data area **does not** contain the value that is required by a VERIFY statement, **all** REPLACE and DELETE operations in the same locate block are inhibited. VERIFY and REPLACE statements that follow the next LOCATE statement are not affected.

Repairing critical catalog table spaces and indexes

An ID with a granted authority receives message `DSNT5001 RESOURCE UNAVAILABLE`, while trying to repair a table space or index in the catalog or directory if table space DSNDB06.SYSDBASE or DSNDB06.SYSUSER is unavailable. If you get this message, you must either make these table spaces available or run the REPAIR

utility on the catalog or directory by using an authorization ID with the installation SYSADM or installation SYSOPR authority.

Updating version information when moving objects to another subsystem

When you move objects that contain system pages from one subsystem to another subsystem, the version information on the target subsystem must match the version information on the source subsystem. If the version information does not match, you cannot access the data on the target subsystem. Follow these steps to move objects to another subsystem and ensure that the version information matches:

1. Ensure that the object definitions on the source and target subsystems are the same. For a table space, each table must have the same number of columns, and each column must be the same data type.

   Recommendation: Use the same ALTER TABLE statement on both the source and target objects.

2. If you are copying indexes that have not been altered in Version 8, check the SYSIBM.SYSINDEXES catalog table on both subsystems to ensure that the value in both the CURRENT_VERSION column and the OLDEST_VERSION column is 0.

3. If the object has been altered since its creation and has never been reorganized, run the REORG utility on the object. You can determine if an object has been altered but not reorganized by checking the values of the OLDEST_VERSION and CURRENT_VERSION columns in SYSIBM.SYSTABLESPACE or SYSIBM.SYSINDEXES. If OLDEST_VERSION is 0 and CURRENT_VERSION is greater than 0, run REORG.

   If you do not own the REORG utility, ensure that you do can insert after the last ALTER to force the creation of a system page.

4. Ensure that enough version numbers are available. For a table space, the combined active number of versions for the object on both the source and target subsystems must be less than 255. For an index, the combined active number of versions must be less than 16. Use the following guidelines to calculate the active number of versions for the object on both the source and target subsystems:

   - If the value in the CURRENT_VERSION column is less than the value in the OLDEST_VERSION column, add the maximum number of versions (255 for a table space or 16 for an index) to the value in the CURRENT_VERSION column.
   - Use the following formula to calculate the number of active versions:

     ```
     number of active_versions =
     MAX(target.CURRENT_VERSION,source.CURRENT_VERSION)
     - MIN(target.OLDEST_VERSION,source.OLDEST_VERSION) + 1
     ```

   If the number of active versions is too high, you must reduce the number of active versions by running REORG on both the source and target objects. Then, use the COPY utility to take a copy, and run MODIFY RECOVERY to recycle the version numbers.

5. Run the DSN1COPY utility with the OBIDXLAT option. On the control statement, specify the proper mapping of table database object identifiers (OBIDs) for the table space or index from the source to the target subsystem.

6. Run REPAIR VERSIONS on the object on the target subsystem. For table spaces, the utility updates the following columns:
   - OLDEST_VERSION and CURRENT_VERSION in SYSTABLEPART
   - VERSION in SYSTABLES

- OLDEST_VERSION and CURRENT_VERSION in SYSTABLESPACE

  For indexes, the utility updates OLDEST_VERSION and CURRENT_VERSION in SYSINDEXES. DB2 uses the following formulas to update these columns in both SYSTABLEPART and SYSINDEXES:

```
CURRENT_VERSION =  MAX(target.CURRENT_VERSION,source.CURRENT_VERSION)
OLDEST_VERSION =  MIN(target.OLDEST_VERSION,source.OLDEST_VERSION)
```

For more information about versions and how they are used by DB2, see Part 2 of *DB2 Administration Guide*.

Terminating or restarting REPAIR

You can terminate a REPAIR job with the TERM UTILITY command. See Chapter 2 of *DB2 Command Reference* for information about TERM UTILITY.

REPAIR cannot be restarted. If you attempt to restart REPAIR, you receive message DSNU191I, which states that the utility cannot be restarted. You must terminate the job with the TERM UTILITY command, and rerun REPAIR from the beginning.

Concurrency and compatibility for REPAIR

DB2 treats individual data and index partitions as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible.

Table 93 shows which claim classes REPAIR drains and any restrictive state that the utility sets on the target object.

Table 93. Claim classes of REPAIR operations

Action	Table space or partition	Index or partition
REPAIR LOCATE KEY DUMP or VERIFY	DW/UTRO	DW/UTRO
REPAIR LOCATE KEY DELETE or REPLACE	DA/UTUT	DA/UTUT
REPAIR LOCATE RID DUMP or VERIFY	DW/UTRO	None
REPAIR LOCATE RID DELETE	DA/UTUT	DA/UTUT
REPAIR LOCATE RID REPLACE	DA/UTUT	None
REPAIR LOCATE TABLESPACE DUMP or VERIFY	DW/UTRO	None
REPAIR LOCATE TABLESPACE REPLACE	DA/UTUT	None
REPAIR LOCATE INDEX PAGE DUMP or VERIFY	None	DW/UTRO
REPAIR LOCATE INDEX PAGE DELETE	None	DA/UTUT

Legend:
- DA - Drain all claim classes - no concurrent SQL access.
- DW - Drain the write claim class - concurrent access for SQL readers.
- UTUT - Utility restrictive state - exclusive control.
- UTRO - Utility restrictive state - read-only access allowed.
- None - Object is not affected by this utility.

REPAIR does not set a utility restrictive state if the target object is DSNDB01.SYSUTILX.

Table 94 and Table 95 on page 521 show which utilities can run concurrently with REPAIR on the same target object. The target object can be a table space, an index space, or a partition of a table space or index space. If compatibility depends on particular options of a utility, that information is also shown in the table.

Table 94 shows which utilities can run concurrently with REPAIR LOCATE by KEY or RID.

Table 94. Utility compatibility with REPAIR, LOCATE by KEY or RID

Utility	DUMP or VERIFY	DELETE or REPLACE
CHECK DATA	No	No
CHECK INDEX	Yes	No
CHECK LOB	Yes	No
COPY INDEXSPACE	Yes	No
COPY TABLESPACE	Yes	No
DIAGNOSE	Yes	Yes
LOAD	No	No
MERGECOPY	Yes	Yes
MODIFY	Yes	Yes
QUIESCE	Yes	No
REBUILD INDEX	No	No
RECOVER INDEX [1]	No	No
RECOVER TABLESPACE	No	No
REORG INDEX [2]	No	No
REORG TABLESPACE UNLOAD CONTINUE or PAUSE	No	No
REORG TABLESPACE UNLOAD ONLY or EXTERNAL	Yes	No
REPAIR DELETE or REPLACE [3]	No	No
REPAIR DUMP or VERIFY	Yes	No
REPORT	Yes	Yes
RUNSTATS INDEX SHRLEVEL CHANGE	Yes	Yes
RUNSTATS INDEX SHRLEVEL REFERENCE	Yes	No
RUNSTATS TABLESPACE	Yes	No
STOSPACE	Yes	Yes
UNLOAD	Yes	No

Notes:

1. REORG INDEX is compatible with LOCATE by RID, DUMP, VERIFY, or REPLACE.
2. RECOVER INDEX is compatible with LOCATE by RID, DUMP, or VERIFY.
3. REPAIR LOCATE INDEX PAGE REPLACE is compatible with LOCATE by RID or REPLACE.

Table 95 shows which utilities can run concurrently with REPAIR LOCATE by PAGE.

Table 95. Utility compatibility with REPAIR, LOCATE by PAGE

Utility or action	TABLESPACE DUMP or VERIFY	TABLESPACE REPLACE	INDEX DUMP or VERIFY	INDEX REPLACE
SQL read	Yes	No	Yes	No
SQL write	No	No	No	No
CHECK DATA	No	No	No	No
CHECK INDEX	Yes	No	Yes	No
CHECK LOB	Yes	No	Yes	No
COPY INDEXSPACE	Yes	Yes	Yes	No
COPY TABLESPACE	Yes	No	Yes	No
DIAGNOSE	Yes	Yes	Yes	Yes
LOAD	No	No	No	No
MERGECOPY	Yes	Yes	Yes	Yes
MODIFY	Yes	Yes	Yes	Yes
QUIESCE	Yes	No	Yes	No
REBUILD INDEX	Yes	No	No	N/A
RECOVER INDEX	Yes	No	No	No
RECOVER TABLESPACE (with no option)	No	No	Yes	Yes
RECOVER TABLESPACE ERROR RANGE	No	No	Yes	Yes
RECOVER TABLESPACE TOCOPY or TORBA	No	No	No	No
REORG INDEX	Yes	Yes	No	No
REORG TABLESPACE UNLOAD CONTINUE or PAUSE	No	No	No	No
REORG TABLESPACE UNLOAD ONLY or EXTERNAL	Yes	No	Yes	Yes
REPAIR DELETE or REPLACE	No	No	No	No
REPAIR DUMP or VERIFY[1]	Yes	No	Yes	No
REPORT	Yes	Yes	Yes	Yes
RUNSTATS INDEX	Yes	Yes	Yes	No
RUNSTATS TABLESPACE	Yes	No	Yes	Yes
STOSPACE	Yes	Yes	Yes	Yes
UNLOAD	Yes	No	Yes	Yes

Notes:

1. REPAIR LOCATE INDEX PAGE REPLACE is compatible with LOCATE TABLESPACE PAGE.

Reviewing REPAIR output

The output from the REPAIR utility can consist of one or more modified pages in the specified DB2 table space or index, and a dump of the contents.

Error messages: At each LOCATE statement, the last data page and the new page that are being located are checked for a few common errors, and messages are issued.

Data checks: Although REPAIR enables you to manipulate both user and DB2 data by bypassing SQL, it does perform some checking of data. For example, if REPAIR tries to write a page with the wrong page number, DB2 abnormally terminates with a 04E code and reason code C200B0. If the page is broken because the broken page bit is on or the incomplete page flag is set, REPAIR issues the following message:

```
DSNU670I + DSNUCBRP - PAGE X'000004' IS A BROKEN PAGE
```

After running REPAIR

CHECK-pending status: You are responsible for violations of referential constraints that are a result of running REPAIR. These violations cause the target table space to be placed in the CHECK-pending status. See Chapter 8, “CHECK DATA,” on page 59 for information about resetting this status.

Sample REPAIR control statements

Example 1: Replacing damaged data and verifying replacement. The following control statement specifies that the REPAIR utility is to perform the following actions:

- Repair the specified page of table space DSN8D81A.DSN8S81D, as indicated by the LOCATE clause.
- Verify that, at the specified offset (50), the damaged data (0A00) is found, as indicated by the VERIFY clause.
- Replace the damaged data with the desired data (0D11), as indicated by the REPLACE clause.
- Initiate a dump beginning at offset 50, for 4 bytes, as indicated by the DUMP clause. You can use the generated dump to verify the replacement.

```
//STEP1    EXEC  DSNUPROC,UID='IUIQU1UH',UTPROC='',SYSTEM='DSN'
//SYSIN DD *
REPAIR OBJECT
  LOCATE TABLESPACE DSN8D81A.DSN8S81D PAGE X'02'
    VERIFY OFFSET 50 DATA X'0A00'
    REPLACE OFFSET 50 DATA X'0D11'
    DUMP OFFSET 50 LENGTH 4
```

Example 2: Removing a nonindexed row that is found by REORG. When reorganizing table space DSNDB04.TS1, assume that you received the following message:

```
DSNU3401 DSNURBXA - ERROR LOADING INDEX, DUPLICATE KEY
                    INDEX = EMPINDEX
                    TABLE = EMP
                    RID OF INDEXED ROW  = X'0000000201'
                    RID OF NONINDEXED ROW = X'0000000503'
```

To resolve this error condition, submit the following control statement, which specifies that REPAIR is to delete the nonindexed row and log the change. (The

LOG keyword is not required; the change is logged by default.) The RID option identifies the row that REPAIR is to delete.

```
REPAIR
  LOCATE TABLESPACE DSNDB04.TS1 RID (X'0000000503')
    DELETE
```

Example 3: Reporting whether catalog and directory DBDs differ. The following control statement specifies that REPAIR is to compare the DBD for DSN8D2AP in the catalog with the DBD for DSN8D2AP in the directory.

```
REPAIR DBD TEST DATABASE DSN8D2AP
```

If the condition code is 0, the DBDs are consistent. If the condition code is not 0, the DBDs might be inconsistent. In this case, run REPAIR DBD with the DIAGNOSE option, as shown in example 4, to find out more detailed information about any inconsistencies.

Example 4: Reporting differences between catalog and directory DBDs. The following control statement specifies that the REPAIR utility is to report information about the inconsistencies between the catalog and directory DBDs for DSN8D2AP. Run this job after you run a REPAIR job with the TEST option (as shown in example 3), and the condition code is not 0. In this example, SYSREC is the output data set, as indicated by the OUTDDN option.

```
REPAIR DBD DIAGNOSE DATABASE DSN8D2AP OUTDDN SYSREC
```

Example 5: Repairing a table space with an orphan row. After running DSN1CHKR on table space SYSDBASE, assume that you receive the following message:

```
DSN1812I ORPHAN ID = 20 ID ENTRY = 0190 FOUND IN
         PAGE = 0000000024
```

From a DSN1PRNT of page X'0000000024' and X'0000002541', you identify that RID X'0000002420' has a forward pointer of X'0000002521'.

Repair the table space by taking the following actions:

1. Submit the following control statement, which specifies that REPAIR is to set the orphan's backward pointer to zeros:

   ```
   REPAIR OBJECT LOG YES
    LOCATE TABLESPACE DSNDB06.SYSDBASE RID X'0000002420'
     VERIFY OFFSET X'0A' DATA X'0000002422'
     REPLACE OFFSET X'0A' DATA X'0000000000'
   ```

   Setting the pointer to zeros prevents the next step from updating link pointers while deleting the orphan. Updating the link pointers can cause DB2 to abnormally terminate if the orphan's pointers are incorrect.
2. Submit the following control statement, which deletes the orphan:

   ```
   REPAIR OBJECT LOG YES
    LOCATE TABLESPACE DSNDB06.SYSDBASE RID X'00002420'
     VERIFY OFFSET X'06' DATA X'00002521'
     DELETE
   ```

Example 6: Resetting restrictive states. The control statement in Figure 91 on page 524 specifies that the REPAIR utility is to reset the following restrictive states for the indicated objects:

- For all indexes on table spaces DBNI1601.TSNI1601 and DBNI1601.TSNI1602, reset RBDP, PSRBDP, or RBDP* status.

- For partition 1 of table space DBNI1601.TSNI1601 and partition 4 of table space DBNI1601.TSNI1602, reset ACHKP status.
- For partitions 1 and 4 of table space DBNI1601.TSNI1601, reset CHKP status.

```
//STEP3    EXEC DSNUPROC,UID='JUNIU116.RECV1',
//            UTPROC='',SYSTEM='SSTR'
//SYSIN    DD *
  REPAIR OBJECT
  SET INDEX (ALL) TABLESPACE DBNI1601.TSNI1601 NORBDPEND
  SET INDEX (ALL) TABLESPACE DBNI1601.TSNI1602 NORBDPEND
  SET TABLESPACE DBNI1601.TSNI1601 PART 1 NOAUXCHKP
  SET TABLESPACE DBNI1601.TSNI1602 PART 4 NOAUXCHKP
  SET TABLESPACE DBNI1601.TSNI1602 PART 1 NOCHECKPEND
  SET TABLESPACE DBNI1601.TSNI1602 PART 4 NOCHECKPEND
/*
```

Figure 91. REPAIR SET example control statement

Example 7: Updating version information. The control statement in Figure 92 specifies that REPAIR is to update the version information in the catalog and directory for table spaces TLKQAST1, TSKQAST2, and TPKQAST3.

```
//STEP1    EXEC DSNUPROC,UID='JUKQU3AS.REPAIR',TIME=1440,
//         UTPROC='',
//         SYSTEM='SSTR',DB2LEV=DB2A
//SYSIN    DD *
  REPAIR VERSIONS TABLESPACE DBKQAST1.TLKQAST1
  REPAIR VERSIONS TABLESPACE DBKQAST2.TSKQAST2
  REPAIR VERSIONS TABLESPACE DBKQAST3.TPKQAST3
```

Figure 92. REPAIR VERSIONS example control statement

Chapter 27. REPORT

The REPORT utility provides information about table spaces. Use REPORT TABLESPACESET to find the names of all the table spaces and tables in a referential structure, including LOB table spaces. The REPORT utility also provides the LOB table spaces that are associated with a base table space. Use REPORT RECOVERY to find information that is necessary for recovering a table space, index, or a table space and all of its indexes.

For a diagram of REPORT syntax and a description of available options, see "Syntax and options of the REPORT control statement" on page 526. For detailed guidance on running this utility, see "Instructions for running REPORT" on page 530.

Output: The output from REPORT TABLESPACESET consists of the names of all table spaces in the table space set that you specify. It also lists all tables in the table spaces and all tables that are dependent on those tables.

The output from REPORT RECOVERY consists of the recovery history from the SYSIBM.SYSCOPY catalog table, log ranges from the SYSIBM.SYSLGRNX directory table, and volume serial numbers where archive log data sets from the BSDS reside. In addition, REPORT RECOVERY output includes information about any indexes on the table space that are in the informational COPY-pending status because this information affects the recoverability of an index. For more information about this situation, see 125.

In a data sharing environment, the REPORT output provides:

- The RBA of when DB2 was migrated to Version 8
- The high and low RBA values of the migrated member
- A list of any SYSLGRNX records from the time before data sharing was enabled that cannot be used to recover to any point in time after data sharing was enabled
- For SYSCOPY, the member from which the image copy was deleted

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:

- RECOVERDB privilege for the database
- DBADM or DBCTRL authority for the database
- SYSCTRL or SYSADM authority

An ID with DBCTRL or DBADM authority over database DSNDB06 can run the REPORT utility on any table space in DSNDB01 (the directory) or DSNDB06 (the catalog), as can any ID with installation SYSOPR, SYSCTRL, or SYSADM authority.

Execution phases of REPORT: The REPORT utility operates in these phases:

Phase	Description
UTILINIT	Performs initialization
REPORT	Collects information
UTILTERM	Performs cleanup

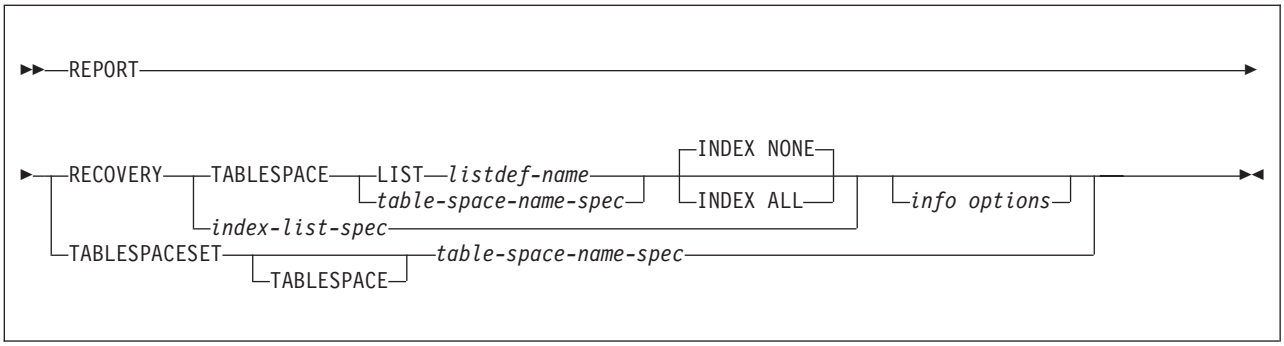
The following topics provide additional information:

- "Syntax and options of the REPORT control statement"
- "Instructions for running REPORT" on page 530
- "Concurrency and compatibility for REPORT" on page 533
- "Reviewing REPORT output" on page 533
- "Sample REPORT control statements" on page 540

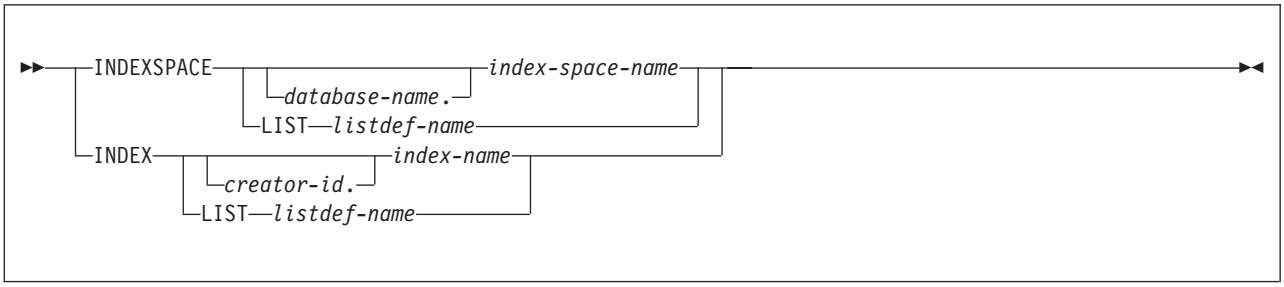
Syntax and options of the REPORT control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram



index-list-spec:



info options:

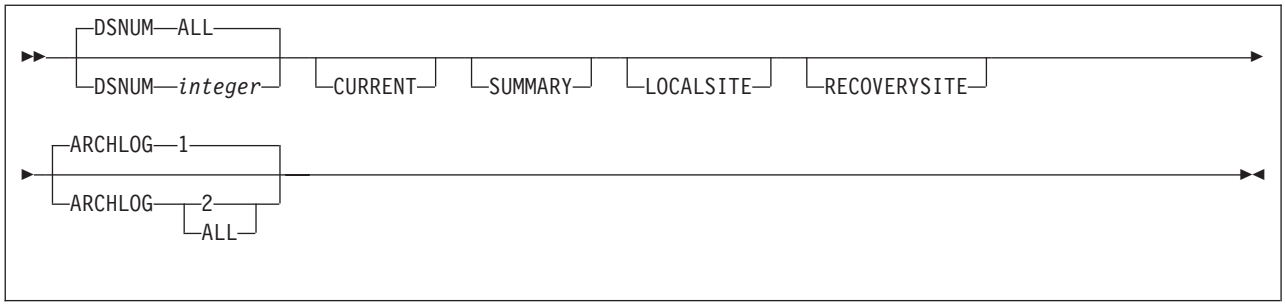
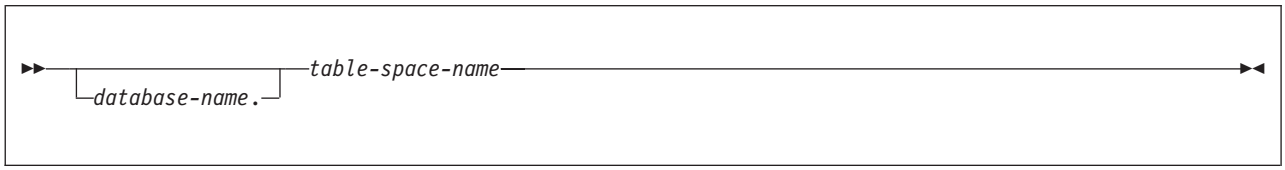


table-space-name-spec:



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

RECOVERY
Indicates that recovery information for the specified table space or index is to be reported.

TABLESPACE *database-name.table-space-name*
For REPORT RECOVERY, specifies the table space (and, optionally, the database to which it belongs) that is being reported.

For REPORT TABLESPACESET, specifies a table space (and, optionally, the database to which it belongs) in the table space set.

database-name
Optionally specifies the database to which the table space belongs.

table-space-name
Specifies the table space.

LIST *listdef-name*

Specifies the name of a previously defined LISTDEF list name. The utility allows one LIST keyword for each control statement of REPORT. The list must contain only table spaces. Do not specify LIST with the TABLESPACE...*table-space-name* specification. The TABLESPACE keyword is required in order to validate the contents of the list. REPORT RECOVERY TABLESPACE is invoked once per item in the list.

For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

INDEXSPACE *database-name.index-space-name*
Specifies the index space that is being reported.

database-name
Optionally specifies the database to which the index space belongs.

index-space-name
Specifies the index space name for the index that is being reported.

LIST *listdef-name*

Specifies the name of a previously defined LISTDEF list name. The utility allows one LIST keyword for each control statement of REPORT. The list must contain only index spaces. Do not specify LIST with the INDEXSPACE...*index-space-name* specification. The INDEXSPACE keyword is required in order to validate the contents of the list. REPORT RECOVERY INDEXSPACE is invoked once for each item in the list.

For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

INDEX *creator-id.index-name*
Specifies the index in the index space that is being reported.

creator-id
Optionally specifies the creator of the index.

index-name
Specifies the index name that is to be reported. Enclose the index name in quotation marks if the name contains a blank.

LIST *listdef-name*

Specifies the name of a previously defined LISTDEF list name. The utility allows one LIST keyword for each control statement of REPORT. The list must contain only index spaces. Do not specify LIST with the INDEX...*index-name* specification. The INDEX keyword is required in order to validate the contents of the list. REPORT RECOVERY INDEX is invoked once for each item in the list.

For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

The following REPORT keywords are optional:

INDEX NONE
Specifies that recovery information for index spaces that are associated with the specified table space is not to be reported.

INDEX ALL
Specifies that recovery information for index spaces that are associated with the specified table space is to be reported.

DSNUM
Identifies a partition or data set for which information is to be reported. Alternatively, DSNUM specifies that information is to be reported for the entire table space or index space.

ALL Specifies that information is to be reported for the entire table space or index space. The **default** is **ALL**.

integer Is the number of a partition or data set for which information is to be reported. The maximum is 4096.

For a partitioned table space or partitioned index space, the integer is its partition number.

For a nonpartitioned table space, find the integer at the end of the data set name, as cataloged in the VSAM catalog. The data set name has the following format:

catname.DSNDB*x*.*dbname*.*tsname*.*y*0001.A*nnn*

In this format:

catname	Is the VSAM catalog name or alias.
x	Is C or D.
dbname	Is the database name.
tsname	Is the table space name.
y	Is I or J.
nnn	Is the data set integer.

CURRENT
Specifies that only the SYSCOPY entries that were written after the last recovery point of the table space are to be reported. The last recovery point is the last full image copy, LOAD REPLACE LOG YES image copy, or REORG LOG YES image copy. If you specify DSNUM ALL, the last recovery point is a full image copy that was taken for the entire table space or index space. However, if you specify the CURRENT option, but the last recovery point does not exist on the active log, DB2 prompts you to mount archive tapes until this point is found.

CURRENT also reports only the SYSLGRNX rows and archive log volumes that were created after the last incremental image copy entry. If no incremental image copies were created, only the SYSLGRNX rows and archive log volumes that were created after the last recovery point are reported.

If you do not specify CURRENT or if no last recovery point exists, all SYSCOPY and SYSLGRNX entries for that table space or index space are reported, including those on archive logs. If you do not specify CURRENT, the entries that were written after the last recovery point are marked with an asterisk (*) in the report.

SUMMARY
Specifies that only a summary of volume serial numbers is to be reported. It reports the following volume serial numbers:
- Where the archive log data sets from the BSDS reside
- Where the image copy data sets from SYSCOPY reside

If you do not specify SUMMARY, recovery information is reported, in addition to the summary of volume serial numbers.

LOCALSITE
Specifies that all SYSCOPY records that were copied from a local site system are to be reported.

RECOVERYSITE
Specifies that all SYSCOPY records that were copied from the recovery site system are to be reported.

ARCHLOG
Specifies which archive log data sets are to be reported.

1 Reports archive log data set 1 only. The **default** is **1**.

2 Reports archive log data set 2 only.

ALL
Reports both archive log data sets 1 and 2.

TABLESPACESET
Indicates that the names of all table spaces in the table space set, as well as the names of all indexes on tables in the table space set, are to be reported.

Instructions for running REPORT

To run REPORT, you must:

1. Prepare the necessary data sets, as described in "Data sets that REPORT uses."
2. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for REPORT, see "Sample REPORT control statements" on page 540.)
3. Prepare a utility control statement that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks."
4. Check the compatibility table in "Concurrency and compatibility for REPORT" on page 533 if you want to run other jobs concurrently on the same target objects.
5. Plan for restart if the REPORT job doesn't complete, as described in "Terminating or restarting REPORT" on page 533.
6. Run REPORT by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Data sets that REPORT uses

Table 96 lists the data sets that REPORT uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 96. Data sets that REPORT uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes

The following object is named in the utility control statement and does not require a DD statement in the JCL:

Table space
Object that is to be reported.

Creating the control statement

Create the utility control statement for the REPORT job. See "Syntax and options of the REPORT control statement" on page 526 for REPORT syntax and option descriptions. See "Sample REPORT control statements" on page 540 for examples of REPORT usage.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in the REPORT utility control statement:

"Reporting recovery information"
"Running REPORT on the catalog and directory" on page 532

Reporting recovery information

You can use the REPORT utility when planning for recovery. REPORT provides information that is necessary for recovering a table space. You can request report information for LOCALSITE, RECOVERYSITE, or both. REPORT RECOVERY displays:

- Recovery information from the SYSIBM.SYSCOPY catalog table, including QUIESCE, COPY, LOAD, REORG, RECOVER TOCOPY, and RECOVER TORBA (or TOLOGPOINT) history. REPORT RECOVERY output also indicates the device type and whether this is the primary or backup copy for LOCALSITE or RECOVERYSITE.
- Log ranges of the table space from the SYSIBM.SYSLGRNX directory.
- Archive log data sets ARCHLOG1, ARCHLOG2, or both from the bootstrap data set.

You can use REPORT TABLESPACESET to find the names of all members of a table space set.

You can also use REPORT to obtain recovery information about the catalog and directory. When doing so, use the CURRENT option to avoid unnecessary mounting of archive tapes.

REPORT uses asterisks to denote any non-COPY entries that it finds in the SYSIBM.SYSCOPY catalog table. For example, an entry that is added by the QUIESCE utility is marked with asterisks in the REPORT output.

Recommendation: For image copies of partitioned table spaces that are taken with the DSNUM ALL option, run REPORT RECOVERY DSNUM ALL. If you run REPORT RECOVERY DSNUM ALL CURRENT, DB2 reports additional historical information that dates back to the last full image copy that was taken for the entire table space.

The REPORT RECOVERY utility output indicates whether any image copies are unusable; image copies that were taken prior to REORG or LOAD events that reset REORG-pending status are marked as unusable. In the REPORT RECOVERY output, look at the IC TYPE and STYPE fields to help you determine which image copies are unusable.

For example, in the sample REPORT RECOVERY output in Figure 93 on page 532, the value in the first IC TYPE field, *R*, indicates that a LOAD REPLACE LOG YES operation occurred. The value in the second IC TYPE field, <F> indicates that a full image copy was taken.

```
DSNU582I  = DSNUPPCP - REPORT RECOVERY TABLESPACE DBKQAA01.TPKQAA01 SYSCOPY ROWS
TIMESTAMP = 2003-02-12-08.37.18.745375,  IC TYPE = *R*,  SHR LVL =  , DSNUM = 0000,
            START LRSN =00000B14E404
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,
            PIT LRSN = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000
JOBNAME  =  T0811104,     AUTHID =   ADMF001 ,     COPYPAGESF = -1.0E+00
NPAGESF = -1.0E+00            ,                        CPAGESF = -1.0E+00
DSNAME    = JUKQU1AA.COPY.STEP1.SYSCOPY                    ,   MEMBER NAME=

TIMESTAMP = 2003-02-12-08.37.56.231114,  IC TYPE = <F>,  SHR LVL = R, DSNUM = 0000,
            START LRSN =00000B283171
DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 000
            PIT LRSN = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000
JOBNAME  =  T0811105,     AUTHID =   ADMF001 ,     COPYPAGESF =  1.63E+02
NPAGESF =  3.6E+02            ,                        CPAGESF =  1.37E+0
DSNAME    = JUKQU1AA.COPY.STEP1.SYSCOPY                    ,   MEMBER NAME =
```

Figure 93. Sample REPORT RECOVERY output before table space placed in REORG-pending status

After this image copy was taken, assume that an event occurred that put the table space in REORG-pending status. Figure 94 shows the next several rows of REPORT RECOVERY output for the same table space. The value in the first ICTYPE field, *X* indicates that a REORG LOG YES event occurred. In the same SYSCOPY record, the value in the STYPE field, A, indicates that this REORG job reset the REORG-pending status. Any image copies that are taken before this status was reset are unusable. (Thus, the full image copy in the REPORT output in Figure 93 is unusable.) The next record contains an F in the IC TYPE field and an X in the STYPE field, which indicates that a full image copy was taken during the REORG job. This image copy is usable.

```
TIMESTAMP = 2003-02-12-08.38.33.366524,  IC TYPE = *X*,  SHR LVL =  , DSNUM = 0000,
            START LRSN =00000B2A6D20
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = A, FILE SEQ = 0000,
            PIT LRSN = 000000000000
LOW DSNUM = 0001,   HIGH DSNUM = 0026,   OLDEST VERSION = 0000,   LOGICAL PART = 0000
JOBNAME  =  T0811108,     AUTHID =   ADMF001 ,     COPYPAGESF = -1.0E+00
NPAGESF = -1.0E+00            ,                        CPAGESF = -1.0E+00
DSNAME    = DBKQAA01.TPKQAA01                              ,   MEMBER NAME =

TIMESTAMP = 2003-02-12-08.38.37.108660,  IC TYPE =  F ,  SHR LVL = R,  DSNUM = 0000,
            START LRSN =00000B386
DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   = X,  FILE SEQ = 0000,
            PIT LRSN = 000000000000
LOW DSNUM = 0001,   HIGH DSNUM = 0026,   OLDEST VERSION = 0000,   LOGICAL PA
JOBNAME  =  T0811108,     AUTHID =   ADMF001 ,     COPYPAGESF =  1.89E+02
NPAGESF =  3.6E+02            ,                        CPAGESF =  3.6E+02
DSNAME    = JUKQU1AA.REORG1.STEP1.SYSCOPY                  ,   MEMBER NAME =
```

Figure 94. Sample REPORT RECOVERY output after REORG-pending status is reset

For a complete explanation of the SYSCOPY fields, see *DB2 SQL Reference*.

Running REPORT on the catalog and directory

REPORT RECOVERY shows the image copies for those table spaces that are not included in SYSIBM.SYSCOPY:

- DSNDB01.SYSUTILX
- DSNDB01.DBD01
- DSNDB06.SYSCOPY

When you execute REPORT RECOVERY on DSNDB01.DBD01, DSNDB01.SYSUTILX, or DSNDB06.SYSCOPY, specify the CURRENT option to avoid unnecessarily mounting archive tapes. If you do not specify CURRENT, DB2 searches for and reports all SYSCOPY records in the log, including those on archive tapes. However, if the CURRENT option is specified and the last recovery point does not exist on the active log, DB2 prompts you to mount archive tapes until this point is found.

You can use REPORT TABLESPACESET on the DB2 catalog and directory table spaces.

Terminating or restarting REPORT

You can terminate a REPORT utility job with the TERM UTILITY command if you have submitted the job or have SYSOPR, SYSCTRL, or SYSADM authority.

You can restart a REPORT utility job, but it starts from the beginning again. For guidance in restarting online utilities, see "Restarting an online utility" on page 43.

Concurrency and compatibility for REPORT

REPORT does not set a utility restrictive state on the target table space or partition.

REPORT can run concurrently on the same target object with any utility or SQL operation.

Reviewing REPORT output

REPORT TABLESPACESET output: The output from REPORT TABLESPACESET consists of the names of all table spaces in the table space set that you specify. It also identifies all tables in the table spaces, and identifies all tables that are dependent on those tables, including LOB table spaces.

For example, the statement REPORT TABLESPACESET TABLESPACE DSN8D81A.DSN8S81D generates the output that is shown in Figure 95 on page 534.

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    DSNUGUTC -  REPORT TABLESPACESET TABLESPACE DSN8D81A.DSN8S81D
DSNU587I  = DSNUPSET - REPORT TABLESPACE SET WITH TABLESPACE DSN8D81A.DSN8S81D

TABLESPACE SET REPORT:

TABLESPACE       : DSN8D81A.DSN8S81D

   TABLE         : DSN8810.DEPT
      INDEXSPACE : DSN8D81A.XDEPT1
           INDEX : DSN8810.XDEPT1
      INDEXSPACE : DSN8D81A.XDEPT2
           INDEX : DSN8810.XDEPT2
      INDEXSPACE : DSN8D81A.XDEPT3
           INDEX : DSN8810.XDEPT3
      DEP  TABLE : DSN8810.DEPT
                   DSN8810.EMP
                   DSN8810.PROJ

TABLESPACE       : DSN8D81A.DSN8S81E

   TABLE         : DSN8810.EMP
      INDEXSPACE : DSN8D81A.XEMP1
           INDEX : DSN8810.XEMP1
      INDEXSPACE : DSN8D81A.XEMP2
           INDEX : DSN8810.XEMP2
      DEP  TABLE : DSN8810.DEPT
                   DSN8810.EMPPROJACT
                   DSN8810.PROJ
TABLESPACE       : DSN8D81A.DSN8S81P

   TABLE         : DSN8810.ACT
      INDEXSPACE : DSN8D81A.XACT1
           INDEX : DSN8810.XACT1
      INDEXSPACE : DSN8D81A.XACT2
           INDEX : DSN8810.XACT2
      DEP  TABLE : DSN8810.PROJACT

   TABLE         : DSN8810.EMPPROJACT
      INDEXSPACE : DSN8D81A.XEMPPROJ
           INDEX : DSN8810.XEMPPROJACT1
      INDEXSPACE : DSN8D81A.KRZC1YHQ
           INDEX : DSN8810.XEMPPROJACT2

   TABLE         : DSN8810.PROJ
      INDEXSPACE : DSN8D81A.XPROJ1
           INDEX : DSN8810.XPROJ1
      INDEXSPACE : DSN8D81A.XPROJ2
           INDEX : DSN8810.XPROJ2
      DEP  TABLE : DSN8810.PROJ
                   DSN8810.PROJACT

   TABLE         : DSN8810.PROJACT
      INDEXSPACE : DSN8D81A.XPROJAC1
           INDEX : DSN8810.XPROJAC1
      DEP  TABLE : DSN8810.EMPPROJACT
DSNU580I    DSNUPORT - REPORT UTILITY COMPLETE - ELAPSED TIME=00:00:00
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 95. Example of REPORT TABLESPACESET output

REPORT RECOVERY output: REPORT RECOVERY displays all information about the image copy data sets and archive log data set that might be required during the recovery.

If the DSVOLSER column of SYSIBM.SYSCOPY is blank, REPORT RECOVERY does not display volume serial numbers for image copy data sets.

The report contains three sections, which include the following types of information:

- Recovery history from the SYSIBM.SYSCOPY catalog table.

  For a description of the fields in the SYSCOPY rows, see the table that describes SYSIBM.SYSCOPY in Appendix D of *DB2 SQL Reference*.
- Log ranges from SYSIBM.SYSLGRNX.
- Volume serial numbers where archive log data sets from the BSDS reside.

If REPORT has no data to display for one or more of these topics, the corresponding sections of the report contain the following message:

```
DSNU588I - NO DATA TO BE REPORTED
```

The RECOVERY ENVIRONMENT RECORD is displayed following message DSNU581I.

SYSIBM.SYSCOPY information is displayed following message DSNU582I. In that section characters are placed around the ″IC TYPE″ value to indicate different unique conditions:

< > - indicates the TABLESPACE image copy preceded an event that reset REORG PENDING. Any point-in-time recovery prior to this point will return the object to REORG PENDING state.

( ) - indicates the TABLESPACE image copy preceded a LOG NO event or the INDEXSPACE image copy preceded an unrecoverable log point. Image copies prior to this point are only usable for point-in-time recoveries.

* * - indicates a full or incremental image copy with none of the preceding special conditions.

Summary information is displayed following message DSNU586I.

The data sharing member, if applicable, is identified in message DSNU592I.

SYSIBM.SYSLGRNX rows are displayed following message DSNU583I.

BSDS information is displayed following message DSNU584I.

Additional information is available in DB2 Messages under each message.

Figure 96 on page 536 shows a sample of REPORT RECOVERY output in a data sharing environment.

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = JUOSU2
DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    DSNUGUTC -  REPORT RECOVERY TABLESPACE DBOS3002.TPOS3002 ARCHLOG ALL
DSNU581I  = DSNUPREC - REPORT RECOVERY TABLESPACE DBOS3002.TPOS3002
    DSNU593I  = DSNUPREC -  REPORT RECOVERY ENVIRONMENT RECORD:
'             MINIMUM   RBA: 000000000000
'             MAXIMUM   RBA: 0000028A5A82
'             MIGRATING RBA: 0000028A5A82
DSNU582I  = DSNUPPCP - REPORT RECOVERY TABLESPACE DBOS3002.TPOS3002 SYSCOPY ROWS
TIMESTAMP = 2002-09-25-08.47.59.979389,  IC TYPE = *R*,  SHR LVL =  ,  DSNUM    = 0000,
   START LRSN =B848B17DBFA1
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,
  PIT LRSN  = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000
JOBNAME  =  T3161102,     AUTHID =   ADMF001 ,     COPYPAGESF = -1.0E+00
NPAGESF = -1.0E+00            ,                        CPAGESF = -1.0E+00
DSNAME    = DBOS3002.TPOS3002                             ,   MEMBER NAME = V81B
TIMESTAMP = 2002-09-25-09.13.27.330574,  IC TYPE =  F ,  SHR LVL = R,  DSNUM    = 0002,
  START LRSN =B848B732F57A
DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,
   PIT LRSN  = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0002
JOBNAME  =  T3161103,     AUTHID =   ADMF001 ,     COPYPAGESF =  1.51E+02
NPAGESF =  1.56E+02           ,                        CPAGESF =  1.5E+02
DSNAME    = JUOSU230.COPY.STEP1.SYSCOPY1                  ,   MEMBER NAME = V81A
TIMESTAMP = 2002-09-25-09.14.02.047456,  IC TYPE =  F ,  SHR LVL = R,  DSNUM    = 0000,
   START LRSN =B848B7505C65
DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,
   PIT LRSN  = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000
JOBNAME  =  T3161103,     AUTHID =   ADMF001 ,     COPYPAGESF =  2.113E+03
NPAGESF =  2.28E+03           ,                        CPAGESF =  1.947E+03
DSNAME    = JUOSU230.COPY.STEP1.SYSCOPY2                  ,   MEMBER NAME = V81A

DSNU586I  = DSNUPSUM - REPORT RECOVERY TABLESPACE DBOS3002.TPOS3002 SUMMARY
DSNU588I  = DSNUPSUM - NO DATA TO BE REPORTED

DSNU592I  = DSNUPREC - REPORT RECOVERY INFORMATION FOR DATA SHARING MEMBER : V81A
DSNU583I  = DSNUPPLR - SYSLGRNX ROWS FROM REPORT RECOVERY FOR TABLESPACE DBOS3002.TPOS3002
```

Figure 96. Example of REPORT RECOVERY in a data sharing environment (Part 1 of 2)

```
UCDATE     UCTIME      START RBA      STOP  RBA     START LRSN     STOP  LRSN    PARTITION   MEMBER ID
092502    08361462   00000DE3C9FA   00000DE5BC88   B848AEDD97F6   B848AF380D15     0001         0001
092502    08361646   00000DE3E91D   00000DE5BD88   B848AEDF5972   B848AF3828FE     0002         0001
092502    08361831   00000DE40804   00000DE5BE88   B848AEE11D01   B848AF382BAF     0003         0001
092502    08362021   00000DE426EB   00000DE5C000   B848AEE2ECAC   B848AF382F61     0004         0001
092502    08362214   00000DE445D2   00000DE5C100   B848AEE4C45C   B848AF3830BC     0005         0001
092502    08362404   00000DE464B9   00000DE5C200   B848AEE69324   B848AF38331C     0006         0001
092502    08362822   00000DE483A0   00000DE5C300   B848AEEA8F55   B848AF3834F7     0007         0001
092502    08363681   00000DE4A2AD   00000DE5C400   B848AEF2C02C   B848AF3836FE     0008         0001
092502    08364286   00000DE4C194   00000DE5C500   B848AEF886EE   B848AF383888     0009         0001
092502    08364946   00000DE4E0A1   00000DE5C600   B848AEFED236   B848AF3839DB     0010         0001
...
092502    08392880   00000DE83A3E   00000DF41788   B848AF96C610   B848B1752630     0015         0001
092502    08392883   00000DE83DA8   00000DF41526   B848AF96CD77   B848B174A721     0016         0001

DSNU584I  = DSNUPPBS - REPORT RECOVERY TABLESPACE DBOS3002.TPOS3002 ARCHLOG1 BSDS VOLUMES
DSNU588I  = DSNUPPBS - NO DATA TO BE REPORTED

DSNU584I  = DSNUPPBS - REPORT RECOVERY TABLESPACE DBOS3002.TPOS3002 ARCHLOG2 BSDS VOLUMES
DSNU588I  = DSNUPPBS - NO DATA TO BE REPORTED

DSNU586I  = DSNUPSUM - REPORT RECOVERY TABLESPACE DBOS3002.TPOS3002 SUMMARY
DSNU588I  = DSNUPSUM - NO DATA TO BE REPORTED

DSNU592I  = DSNUPREC - REPORT RECOVERY INFORMATION FOR DATA SHARING MEMBER : V81B
DSNU583I  = DSNUPPLR - SYSLGRNX ROWS FROM REPORT RECOVERY FOR TABLESPACE DBOS3002.TPOS3002
UCDATE     UCTIME      START RBA      STOP  RBA     START LRSN     STOP  LRSN    PARTITION   MEMBER ID
092502    08502167   00000010981C   000000128AAB   B848B20565F4   B848B206C7D0     0001         0002
092502    08502170   00000010B7CF   000000128BAB   B848B2056F2B   B848B206C976     0002         0002
092502    08502176   00000010D6B6   000000128CAB   B848B2057C32   B848B206CAED     0003         0002
092502    08502182   00000010F5DA   000000128DAB   B848B2058BE5   B848B206CC55     0004         0002
092502    08502188   0000001115CF   000000128EAB   B848B2059AAD   B848B206CDCA     0005         0002
092502    08502193   0000001134B6   000000129000   B848B205A5C3   B848B206CF53     0006         0002
...
092502    09064089   00000083C29F   0000009A2DB2   B848B5AB422E   B848B6D42EBD     0015         0002
092502    09070293   00000089949A   0000009A3090   B848B5C04679   B848B6D4DEE7     0016         0002
DSNU584I  = DSNUPPBS - REPORT RECOVERY TABLESPACE DBOS3002.TPOS3002 ARCHLOG1 BSDS VOLUMES
DSNU588I  = DSNUPPBS - NO DATA TO BE REPORTED
DSNU584I  = DSNUPPBS - REPORT RECOVERY TABLESPACE DBOS3002.TPOS3002 ARCHLOG2 BSDS VOLUMES
DSNU588I  = DSNUPPBS - NO DATA TO BE REPORTED

DSNU586I  = DSNUPSUM - REPORT RECOVERY TABLESPACE DBOS3002.TPOS3002 SUMMARY
DSNU588I  = DSNUPSUM - NO DATA TO BE REPORTED
DSNU589I  = DSNUPREC - REPORT RECOVERY TABLESPACE DBOS3002.TPOS3002 COMPLETE

DSNU580I    DSNUPORT - REPORT UTILITY COMPLETE - ELAPSED TIME=00:00:01
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
NO CONTROL STATEMENT PROVIDED.
```

Figure 96. Example of REPORT RECOVERY in a data sharing environment (Part 2 of 2)

Figure 97 on page 538 shows sample output for the statement REPORT RECOVERY TABLESPACE ARCHLOG. Under message DSNU584I, the archive log entries after the last recovery point are marked with an asterisk (*). If you code the CURRENT option, the output from message DSNU584I would include only the archive logs after the last recovery point and the asterisk (*) would not be included in the report.

REPORT

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = D7058005.RCVR3
DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    DSNUGUTC -  REPORT RECOVERY TABLESPACE DB580501.TS580501 ARCHLOG ALL
DSNU581I  = DSNUPREC - REPORT RECOVERY TABLESPACE DB580501.TS580501
DSNU593I  = DSNUPREC -  REPORT RECOVERY ENVIRONMENT RECORD:
'             MINIMUM   RBA: 000000000000
'             MAXIMUM   RBA: FFFFFFFFFFFF
'             MIGRATING RBA: 000000000000
   DSNU582I  = DSNUPPCP - REPORT RECOVERY TABLESPACE DB580501.TS580501 SYSCOPY ROWS
TIMESTAMP = 2002-09-17-10.03.16.784238,  IC TYPE = *Q*,  SHR LVL =  ,  DSNUM    = 0000,
   START LRSN =00001E58E60D
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = W,  FILE SEQ = 0000,
   PIT LRSN  = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 3648
JOBNAME  =  T3951105,     AUTHID =   ADMF001 ,     COPYPAGESF = -1.0E+00
NPAGESF = -1.0E+00            ,                       CPAGESF = -1.0E+00
DSNAME    = DB580501.TS580501                           ,   MEMBER NAME =

TIMESTAMP = 2002-09-17-10.03.22.937931,  IC TYPE = *Z*,  SHR LVL =  ,  DSNUM    = 0000,
   START LRSN =00001E5956A3
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,
   PIT LRSN  = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000
JOBNAME  =  T3951105,     AUTHID =   ADMF001 ,     COPYPAGESF = -1.0E+00
NPAGESF = -1.0E+00            ,                       CPAGESF = -1.0E+00
DSNAME    = DB580501.TS580501                           ,   MEMBER NAME =

TIMESTAMP = 2002-09-17-10.03.43.118193,  IC TYPE = *Q*,  SHR LVL =  ,  DSNUM    = 0000,
   START LRSN =00001E5A7B9D
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = W,  FILE SEQ = 0000,
   PIT LRSN  = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 3648
JOBNAME  =  T3951106,     AUTHID =   ADMF001 ,     COPYPAGESF = -1.0E+00
NPAGESF = -1.0E+00            ,                       CPAGESF = -1.0E+00
DSNAME    = DB580501.TS580501                           ,   MEMBER NAME =

TIMESTAMP = 2002-09-17-10.03.53.881540,  IC TYPE = *Z*,  SHR LVL =  ,  DSNUM    = 0000,
   START LRSN =00001E5ADC6E
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,
   PIT LRSN  = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000
JOBNAME  =  T3951106,     AUTHID =   ADMF001 ,     COPYPAGESF = -1.0E+00
NPAGESF = -1.0E+00            ,                       CPAGESF = -1.0E+00
DSNAME    = DB580501.TS580501                           ,   MEMBER NAME =

TIMESTAMP = 2002-09-17-10.04.02.955333,  IC TYPE = *Q*,  SHR LVL =  ,  DSNUM    = 0000,
   START LRSN =00001E624A3C
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = W,  FILE SEQ = 0000,
   PIT LRSN  = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 3648
JOBNAME  =  T3951106,     AUTHID =   ADMF001 ,     COPYPAGESF = -1.0E+00
NPAGESF = -1.0E+00            ,                       CPAGESF = -1.0E+00
DSNAME    = DB580501.TS580501                           ,   MEMBER NAME =
DSNU586I  = DSNUPSUM - REPORT RECOVERY TABLESPACE DB580501.TS580501 SUMMARY
DSNU588I  = DSNUPSUM - NO DATA TO BE REPORTED

DSNU583I  = DSNUPPLR - SYSLGRNX ROWS FROM REPORT RECOVERY FOR TABLESPACE DB580501.TS580501
UCDATE     UCTIME      START RBA      STOP  RBA     START LRSN     STOP  LRSN    PARTITION   MEMBER ID
091702    10025977   00001E4FD319   00001E4FEB91   00001E4FD319   00001E4FEB91     0000         0000     *
091702    10030124   00001E505B93   00001E58BC23   00001E505B93   00001E58BC23     0000         0000     *
091702    10032302   00001E59A637   00001E5A5258   00001E59A637   00001E5A5258     0000         0000     *
091702    10035391   00001E5B26AB   00001E6222F3   00001E5B26AB   00001E6222F3     0000         0000     *
```

Figure 97. Example of REPORT RECOVERY TABLESPACE ARCHLOG (Part 1 of 3)

```
DSNU584I  = DSNUPPBS - REPORT RECOVERY TABLESPACE DB580501.TS580501 ARCHLOG1 BSDS VOLUMES
START TIME       END TIME    START RBA      END RBA    UNIT  VOLSER   DATA SET NAME
20022601702454 20022601704156 00001E48B000 00001E629FFF SYSDA  SCR03  DSNC810.ARCHLOG1.A0000005     *
DSNU584I  = DSNUPPBS - REPORT RECOVERY TABLESPACE DB580501.TS580501 ARCHLOG2 BSDS VOLUMES
DSNU588I  = DSNUPPBS - NO DATA TO BE REPORTED
DSNU586I  = DSNUPSUM - REPORT RECOVERY TABLESPACE DB580501.TS580501 SUMMARY
                ARCHLOG1 BSDS VOLSER(S)            SCR03  *
       DSNU589I  = DSNUPREC - REPORT RECOVERY TABLESPACE DB580501.TS580501 COMPLETE

DSNU580I    DSNUPORT - REPORT UTILITY COMPLETE - ELAPSED TIME=00:00:00
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = D7058005.RCVR3
DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    DSNUGUTC -  REPORT RECOVERY TABLESPACE DB580501.TS580501 CURRENT
DSNU581I  = DSNUPREC - REPORT RECOVERY TABLESPACE DB580501.TS580501
DSNU585I  = DSNUPREC - REPORT RECOVERY TABLESPACE DB580501.TS580501 CURRENT
DSNU593I  = DSNUPREC -  REPORT RECOVERY ENVIRONMENT RECORD:
'             MINIMUM   RBA: 000000000000
'             MAXIMUM   RBA: FFFFFFFFFFFF
'             MIGRATING RBA: 000000000000
      DSNU582I  = DSNUPPCP - REPORT RECOVERY TABLESPACE DB580501.TS580501 SYSCOPY ROWS
TIMESTAMP = 2002-09-17-10.03.16.784238,  IC TYPE = *Q*,  SHR LVL =  ,  DSNUM    = 0000,
   START LRSN =00001E58E60D
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = W,  FILE SEQ = 0000,
   PIT LRSN  = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 3648
JOBNAME  =  T3951105,     AUTHID =   ADMF001 ,     COPYPAGESF = -1.0E+00
NPAGESF = -1.0E+00            ,                       CPAGESF = -1.0E+00
DSNAME    = DB580501.TS580501                           ,   MEMBER NAME =

TIMESTAMP = 2002-09-17-10.03.22.937931,  IC TYPE = *Z*,  SHR LVL =  ,  DSNUM    = 0000,
   START LRSN =00001E5956A3
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,
   PIT LRSN  = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000
JOBNAME  =  T3951105,     AUTHID =   ADMF001 ,     COPYPAGESF = -1.0E+00
NPAGESF = -1.0E+00            ,                       CPAGESF = -1.0E+00
DSNAME    = DB580501.TS580501                           ,   MEMBER NAME =
...

TIMESTAMP = 2002-09-17-10.04.02.955333,  IC TYPE = *Q*,  SHR LVL =  ,  DSNUM    = 0000,
   START LRSN =00001E624A3C
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = W,  FILE SEQ = 0000,
   PIT LRSN  = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 3648
JOBNAME  =  T3951106,     AUTHID =   ADMF001 ,     COPYPAGESF = -1.0E+00
NPAGESF = -1.0E+00            ,                       CPAGESF = -1.0E+00
DSNAME    = DB580501.TS580501                           ,   MEMBER NAME =

DSNU586I  = DSNUPSUM - REPORT RECOVERY TABLESPACE DB580501.TS580501 SUMMARY
DSNU588I  = DSNUPSUM - NO DATA TO BE REPORTED

DSNU583I  = DSNUPPLR - SYSLGRNX ROWS FROM REPORT RECOVERY FOR TABLESPACE DB580501.TS580501
UCDATE     UCTIME      START RBA      STOP  RBA     START LRSN     STOP  LRSN    PARTITION   MEMBER ID
091702    10025977   00001E4FD319   00001E4FEB91   00001E4FD319   00001E4FEB91     0000         0000
091702    10030124   00001E505B93   00001E58BC23   00001E505B93   00001E58BC23     0000         0000
091702    10032302   00001E59A637   00001E5A5258   00001E59A637   00001E5A5258     0000         0000
091702    10035391   00001E5B26AB   00001E6222F3   00001E5B26AB   00001E6222F3     0000         0000

DSNU584I  = DSNUPPBS - REPORT RECOVERY TABLESPACE DB580501.TS580501 ARCHLOG1 BSDS VOLUMES
START TIME       END TIME    START RBA      END RBA    UNIT  VOLSER   DATA SET NAME
20022601702454 20022601704156 00001E48B000 00001E629FFF SYSDA  SCR03  DSNC810.ARCHLOG1.A0000005
```

Figure 97. Example of REPORT RECOVERY TABLESPACE ARCHLOG (Part 2 of 3)

```
DSNU586I  = DSNUPSUM - REPORT RECOVERY TABLESPACE DB580501.TS580501 SUMMARY
               ARCHLOG1 BSDS VOLSER(S)            SCR03
DSNU589I  = DSNUPREC - REPORT RECOVERY TABLESPACE DB580501.TS580501 COMPLETE
DSNU580I    DSNUPORT - REPORT UTILITY COMPLETE - ELAPSED TIME=00:00:00
DSNU050I    DSNUGUTC -  RECOVER TABLESPACE DB580501.TS580501 LOGONLY
DSNU532I  = DSNUCALA - RECOVER TABLESPACE DB580501.TS580501   START
DSNU549I  = DSNUCALA - RECOVER TABLESPACE DB580501.TS580501
                        USES ONLY DB2 LOGS STARTING FROM LOGPOINT=X'00001E5A5258'.
DSNU513I  = DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 00001E5B26AB LRSN 00001E5B26AB TO
                                                          RBA 00001E6222F3 LRSN 00001E6222F3
DSNU500I    DSNUCBDR - RECOVERY COMPLETE, ELAPSED TIME=00:00:03
DSNU050I    DSNUGUTC -  REBUILD INDEX(ALL) TABLESPACE DB580501.TS580501
DSNU555I  = DSNUCRUL - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS PROCESSED=45
DSNU705I    DSNUCRIB - UNLOAD PHASE COMPLETE - ELAPSED TIME=00:00:00
DSNU394I  = DSNURBXC - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=25 FOR INDEX ADMF001.IX580501
DSNU394I  = DSNURBXC - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=20 FOR INDEX ADMF001.IX580502
DSNU391I    DSNUCRIB - SORTBLD PHASE STATISTICS. NUMBER OF INDEXES = 2
DSNU392I    DSNUCRIB - SORTBLD PHASE COMPLETE, ELAPSED TIME = 00:00:09
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 97. Example of REPORT RECOVERY TABLESPACE ARCHLOG (Part 3 of 3)

Sample REPORT control statements

Example 1: Reporting recovery information for a table space. The following control statement specifies that the REPORT utility is to provide recovery information for table space DSN8D81A.DSN8S81E.

```
//STEP1    EXEC DSNUPROC,UID='IUKUU206.REPORT2',
//            UTPROC='',SYSTEM='DSN'
//SYSIN    DD *
REPORT RECOVERY
       TABLESPACE DSN8D81A.DSN8S81E
//*
```

The preceding statement produces output similar to the output shown in Figure 98 on page 541.

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    DSNUGUTC -  REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E
DSNU581I  = DSNUPREC - REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E
DSNU593I  = DSNUPREC -  REPORT RECOVERY ENVIRONMENT RECORD:
'             MINIMUM   RBA: 000000000000
'             MAXIMUM   RBA: FFFFFFFFFFFF
'             MIGRATING RBA: 000000000000
DSNU582I  = DSNUPPCP - REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E SYSCOPY ROWS
TIMESTAMP = 2002-11-14-14.58.46.843369, IC TYPE = *Z*, SHR LVL =  , DSNUM    = 0004,
              START LRSN=0000018C3750
DEV TYPE  =         ,                 IC BACK =  , STYPE   =  , FILE SEQ =0000,
              PIT LRSN = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 3647
JOBNAME  =  DSNTEJ1 ,     AUTHID =   SYSADM  ,     COPYPAGESF = -1.0E+00
NPAGESF = -1.0E+00           ,                      CPAGESF = -1.0E+00
DSNAME    = DSN8D81A.DSN8S81E                          ,   MEMBER NAME =

TIMESTAMP = 2002-11-14-14.58.46.843369, IC TYPE = *Z*,  SHR LVL =  ,  DSNUM = 0003,
              START LRSN=0000018C3750
DEV TYPE  =         ,                   IC BACK =  , STYPE   =  FILE SEQ =0000,
              PIT LRSN = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0001,   LOGICAL PART = 0001
JOBNAME  =  DSNTEJ1 ,     AUTHID =   SYSADM  ,     COPYPAGESF = -1.0E+00
NPAGESF = -1.0E+00           ,                      CPAGESF = -1.0E+00
DSNAME    = DSN8D81A.DSN8S81E                          ,   MEMBER NAME =

...
DSNU586I  = DSNUPSUM - REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E SUMMARY
DSNU588I  = DSNUPSUM - NO DATA TO BE REPORTED

DSNU583I  = DSNUPPLR - SYSLGRNX ROWS FROM REPORT RECOVERY FOR TABLESPACE DSN8D8
 UCDATE   UCTIME     START RBA     STOP  RBA    START LRSN    STOP  LRSN  PARTITION  MEMBER ID
 111402  14582223  00000173BEAA  000001742B91  00000173BEAA  000001742B91   0001        0000
 111402  14582235  00000173DDD0  000001742B91  00000173DDD0  000001742B91   0002        0000
 111402  14582245  00000173FCF0  000001742B91  00000173FCF0  000001742B91   0003        0000
 111402  14582255  000001741BF8  000001742B91  000001741BF8  000001742B91   0004        0000
 111402  14582696  00000177E5DB  0000017F890B  00000177E5DB  0000017F890B   0001        0000
 111402  14582698  00000177E98B  00000182D2B8  00000177E98B  00000182D2B8   0002        0000
 111402  14582699  00000177ECFF  0000017B05E4  00000177ECFF  0000017B05E4   0003        0000
 111402  14582699  00000177F09E  0000018798B8  00000177F09E  0000018798B8   0004        0000
 111402  14584686  0000018C97A8  0000019451CB  0000018C97A8  0000019451CB   0001        0000
 111402  14584725  0000018DAB26  00000194543B  0000018DAB26  00000194543B   0002        0000
 111402  14584755  0000018EBEC5  0000019456AB  0000018EBEC5  0000019456AB   0004        0000
 111402  14591633  000001AFB000  000001AFD033  000001AFB000  000001AFD033   0003        0000
 111402  14591706  000001AFE4AA  000001B0D1AF  000001AFE4AA  000001B0D1AF   0003        0000
 111402  14592109  000001B2EADA  000001B39F0D  000001B2EADA  000001B39F0D   0003        0000

DSNU584I  = DSNUPPBS - REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E ARCHLOG1 BS
DSNU588I  = DSNUPPBS - NO DATA TO BE REPORTED
DSNU586I  = DSNUPSUM - REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E SUMMARY
DSNU588I  = DSNUPSUM - NO DATA TO BE REPORTED
DSNU589I  = DSNUPREC - REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E COMPLETE

DSNU580I    DSNUPORT - REPORT UTILITY COMPLETE - ELAPSED TIME=00:00:00
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 98. Example output for REPORT RECOVERY

Example 2: Reporting table spaces that are referentially related. The following control statement specifies that REPORT is to provide a list of all table spaces that are referentially related to table space DSN8D81A.DSN8S81E. The output also includes a list of any related LOB table spaces and of all indexes on the tables in those table spaces.

```
REPORT TABLESPACESET TABLESPACE DSN8D81A.DSN8S81E
```

The preceding statement produces output similar to the output shown in Figure 99.

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    DSNUGUTC -  REPORT TABLESPACESET TABLESPACE DSN8D81A.DSN8S81E
DSNU587I  = DSNUPSET - REPORT TABLESPACE SET WITH TABLESPACE DSN8D81A.DSN8S81E

TABLESPACE SET REPORT:

TABLESPACE       : DSN8D81A.DSN8S81D

   TABLE         : DSN8810.DEPT
      INDEXSPACE : DSN8D81A.XDEPT1
           INDEX : DSN8810.XDEPT1
      INDEXSPACE : DSN8D81A.XDEPT2
           INDEX : DSN8810.XDEPT2
      INDEXSPACE : DSN8D81A.XDEPT3
           INDEX : DSN8810.XDEPT3
      DEP  TABLE : DSN8810.DEPT
                   DSN8810.EMP
                   DSN8810.PROJ

TABLESPACE       : DSN8D81A.DSN8S81E

   TABLE         : DSN8810.EMP
      INDEXSPACE : DSN8D81A.XEMP1
           INDEX : DSN8810.XEMP1
      INDEXSPACE : DSN8D81A.XEMP2
           INDEX : DSN8810.XEMP2
      DEP  TABLE : DSN8810.DEPT
                   DSN8810.EMPPROJACT
                   DSN8810.PROJ
TABLESPACE       : DSN8D81A.DSN8S81P

   TABLE         : DSN8810.ACT
      INDEXSPACE : DSN8D81A.XACT1
           INDEX : DSN8810.XACT1
      INDEXSPACE : DSN8D81A.XACT2
           INDEX : DSN8810.XACT2
      DEP  TABLE : DSN8810.PROJACT

   TABLE         : DSN8810.EMPPROJACT
      INDEXSPACE : DSN8D81A.XEMPPROJ
           INDEX : DSN8810.XEMPPROJACT1
      INDEXSPACE : DSN8D81A.KRZC1YHQ
           INDEX : DSN8810.XEMPPROJACT2

   TABLE         : DSN8810.PROJ
      INDEXSPACE : DSN8D81A.XPROJ1
           INDEX : DSN8810.XPROJ1
      INDEXSPACE : DSN8D81A.XPROJ2
           INDEX : DSN8810.XPROJ2
      DEP  TABLE : DSN8810.PROJ
                   DSN8810.PROJACT

   TABLE         : DSN8810.PROJACT
      INDEXSPACE : DSN8D81A.XPROJAC1
           INDEX : DSN8810.XPROJAC1
      DEP  TABLE : DSN8810.EMPPROJACT
DSNU580I    DSNUPORT - REPORT UTILITY COMPLETE - ELAPSED TIME=00:00:00
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 99. Example output for REPORT TABLESPACESET

Example 3: Reporting recovery information for a partition of a partitioned table space. The following control statement specifies that REPORT is to provide recovery information for partition 4 of table space DSN8D81A.DSN8S81E. The partition number is indicated by the DSNUM option.

```
REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E DSNUM 4
```

The preceding statement produces output similar to the output shown in Figure 100.

```
1DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    DSNUGUTC -  REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E DSNUM 4
 DSNU581I  = DSNUPREC - REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E
 DSNU593I  = DSNUPREC -  REPORT RECOVERY ENVIRONMENT RECORD:
 '             MINIMUM   RBA: 000000000000
 '             MAXIMUM   RBA: FFFFFFFFFFFF
 '             MIGRATING RBA: 000000000000
 DSNU582I  = DSNUPPCP - REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E SYSCOPY ROW
TIMESTAMP = 2002-11-14-14.58.46.843369, IC TYPE = *Z*, SHR LVL =  , DSNUM    = 0004,
            START LRSN=0000018C3750
DEV TYPE  =         ,                  IC BACK =  , STYPE   =  , FILE SEQ =0000,
            PIT LRSN = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,  OLDEST VERSION = 0000, LOGICAL PART = 3647
JOBNAME  =  DSNTEJ1 ,     AUTHID =   SYSADM  ,     COPYPAGESF = -1.0E+00
NPAGESF = -1.0E+00            ,                       CPAGESF = -1.0E+00
DSNAME    = DSN8D81A.DSN8S81E                            ,   MEMBER NAME =
...
DSNU586I  = DSNUPSUM - REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E SUMMARY
DSNU588I  = DSNUPSUM - NO DATA TO BE REPORTED

DSNU583I  = DSNUPPLR - SYSLGRNX ROWS FROM REPORT RECOVERY FOR TABLESPACE DSN8D8
 UCDATE  UCTIME     START RBA     STOP  RBA    START LRSN    STOP LRSN  PARTITION  MEMBER ID
 111402 14582255  000001741BF8  000001742B91  000001741BF8  000001742B91   0004      0000
 111402 14582699  00000177F09E  0000018798B8  00000177F09E  0000018798B8   0004      0000
 111402 14584755  0000018EBEC5  0000019456AB  0000018EBEC5  0000019456AB   0004      0000

DSNU584I  = DSNUPPBS - REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E ARCHLOG1 BS
DSNU588I  = DSNUPPBS - NO DATA TO BE REPORTED

DSNU586I  = DSNUPSUM - REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E SUMMARY
DSNU588I  = DSNUPSUM - NO DATA TO BE REPORTED
DSNU589I  = DSNUPREC - REPORT RECOVERY TABLESPACE DSN8D81A.DSN8S81E COMPLETE

DSNU580I    DSNUPORT - REPORT UTILITY COMPLETE - ELAPSED TIME=00:00:00
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 100. Example output for REPORT RECOVERY DSNUM

Example 4: Reporting recovery information for an index. The control statement in Figure 101 specifies that REPORT is to provide recovery information for index DSN8810.XDEPT1.

The preceding statement produces output similar to the output shown in

```
//STEP3    EXEC DSNUPROC,UID='IUJMU111.REPORT',
//         UTPROC='',
//         SYSTEM='SSTR',DB2LEV=DB2A
//SYSIN    DD *
    REPORT RECOVERY INDEX DSN8810.XDEPT1
/*
```

Figure 101. Example REPORT RECOVERY statement for an index

Figure 102 on page 544.

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    DSNUGUTC -  REPORT RECOVERY INDEX DSN8810.XDEPT1
DSNU581I  = DSNUPREC - REPORT RECOVERY INDEX DSN8810.XDEPT1
DSNU593I  = DSNUPREC -  REPORT RECOVERY ENVIRONMENT RECORD:
'             MINIMUM   RBA: 000000000000
'             MAXIMUM   RBA: FFFFFFFFFFFF
'             MIGRATING RBA: 000000000000
DSNU582I  = DSNUPPCP - REPORT RECOVERY INDEX DSN8810.XDEPT1 SYSCOPY ROWS
TIMESTAMP = 2003-03-18-13.53.09.802224,  IC TYPE =  F , SHR LVL = R,  DSNUM = 0000,
            START LRSN =000004FE4537
DEV TYPE  = 3390    ,                    IC BACK =   ,  STYPE  =  , FILE SEQ = 0000,
            PIT LRSN = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000
JOBNAME  =  ADMF001A,     AUTHID =   ADMF001 ,     COPYPAGESF =  5.0E+00
NPAGESF =  1.2E+01            ,                        CPAGESF =  0.0E0
DSNAME    = ADMF001.LOCAL                                 ,   MEMBER NAME =

DSNU586I  = DSNUPSUM - REPORT RECOVERY INDEX DSN8810.XDEPT1 SUMMARY
DSNU588I  = DSNUPSUM - NO DATA TO BE REPORTED

DSNU583I  = DSNUPPLR - SYSLGRNX ROWS FROM REPORT RECOVERY FOR INDEX DSN8810.XDEP
DSNU588I  = DSNUPPLR - NO DATA TO BE REPORTED
```

Figure 102. Example output for REPORT RECOVERY INDEX

Chapter 28. RESTORE SYSTEM

The RESTORE SYSTEM utility invokes z/OS DFSMShsm (Version 1 Release 5 or above) to recover a DB2 subsystem or a data sharing group to a previous point in time. To perform the recovery, the utility uses data that is copied by the BACKUP SYSTEM utility. All data sets that you want to recover must be SMS-managed data sets.

The RESTORE SYSTEM utility can be run from any member in a data sharing group, even one that is normally quiesced when any backups are taken. Any member in the data sharing group that is active at or beyond the log truncation point must be restarted, and its logs are truncated to the SYSPITR LRSN point. You can specify the SYSPITR LRSN point in the CRESTART control statement of the DSNJU003 (Change Log Inventory) utility. Any data sharing group member that is normally quiesced at the time the backups are taken and is not active at or beyond the log truncation point does not need to be restarted.

Restrictions: DFSMShsm V1R5 can maintain multiple backup versions of copy pools. However, you cannot specify a particular backup version to be used by the RESTORE SYSTEM utility. RESTORE SYSTEM uses the latest version before the log truncation point. You can specify the log truncation point with the CRESTART SYSPITR option of the DSNJU003 (Change Log Inventory) stand-alone utility. For more information about this option, see Chapter 36, “DSNJU003 (change log inventory),” on page 677. For information about copy pools and associated backup storage groups, see *z/OS DFSMSdfp Storage Administration Reference*.

RESTORE SYSTEM does not restore logs; the utility only applies the logs. If you specified BACKUP SYSTEM FULL to create copies of both the data and the logs, you can restore the logs by another method. For more information about BACKUP SYSTEM FULL, see Chapter 5, “BACKUP SYSTEM,” on page 49.

Output: Output for RESTORE SYSTEM is the recovered copy of the data volume or volumes.

Related information: For more information about the use of RESTORE SYSTEM in system level point-in-time recovery, see Part 4 of *DB2 Administration Guide*.

Authorization required: To run this utility, you must use a privilege set that includes Installation SYSADM authority.

Execution phases of RESTORE SYSTEM: The RESTORE SYSTEM utility operates in the following phases:

Phase	Description
UTILINIT	Performs initialization and setup
RESTORE	Locates and restores the volume copies if the LOGONLY option is not specified
LOGAPPLY	Applies the outstanding log changes to the database
UTILTERM	Performs cleanup

The following topics provide additional information:

Syntax and options of the RESTORE SYSTEM control statement

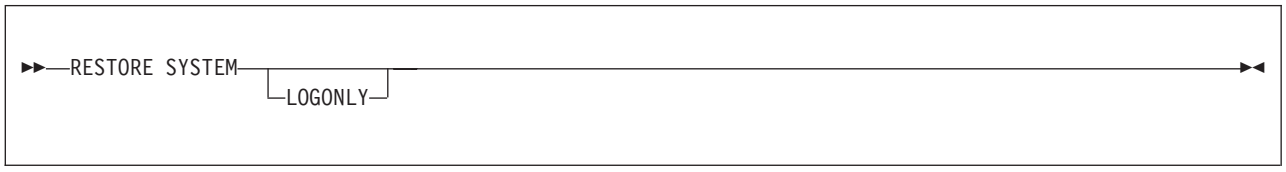
The utility control statement defines the function that the utility job performs. Use the ISPF/PDF edit function to create a control statement and to save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

When you specify RESTORE SYSTEM, you can specify only the following statements in the same step:

- DIAGNOSE
- OPTIONS PREVIEW
- OPTIONS OFF
- OPTIONS KEY
- OPTIONS EVENT WARNING

In addition, RESTORE SYSTEM must be the last statement in SYSIN.

Syntax diagram



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

LOGONLY
Specifies that the database volumes have already been restored, so the RESTORE phase is skipped. Use this option when the database volumes have already been restored outside of DB2. If the subsystem is at a tracker site, you must specify the LOGONLY option. For more information about using a tracker site, see Part 4 (Volume 1) of *DB2 Administration Guide*.

By default, RESTORE SYSTEM recovers the data from the database copy pool during the RESTORE phase and then applies logs to the point in time at which the existing logs were truncated during the LOGAPPLY phase. The RESTORE utility never restores logs from the log copy pool.

Instructions for running RESTORE SYSTEM

To run RESTORE SYSTEM, you must:

1. Read "Before running RESTORE SYSTEM" on page 547.

2. Prepare the necessary data sets, as described in "Data sets that RESTORE SYSTEM uses."
3. Create JCL statements by either "Using the supplied JCL procedure (DSNUPROC)" on page 34 or "Creating the JCL data set yourself by using the EXEC statement" on page 37.
4. Prepare a utility control statement that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks."
5. Check "Concurrency and compatibility for RESTORE SYSTEM" on page 548 if you want to run other jobs concurrently on the same target objects.
6. Plan for restarting RESTORE SYSTEM if the job doesn't complete, as described in "Terminating and restarting RESTORE SYSTEM" on page 548.
7. Run RESTORE SYSTEM by either "Using the supplied JCL procedure (DSNUPROC)" on page 34 or "Creating the JCL data set yourself by using the EXEC statement" on page 37.

Before running RESTORE SYSTEM

Complete the following steps prior to running RESTORE SYSTEM:

1. Stop DB2.
2. Run DSNJU003 (Change Log Inventory) with the CRESTART SYSPITR option. For SYSPITR, specify the log truncation point that corresponds to the previous point in time to which the system is to be recovered. The utility allows SYSPITR only after new-function mode is enabled.
3. Start DB2. When the restart that is specified by CRESTART SYSPITR completes, DB2 enters system RECOVER-pending and access maintenance mode. During system RECOVER-pending mode, you can run only the RESTORE SYSTEM utility.
4. Ensure that the ICF catalog volumes for DB2 data are not active. The ICF catalog for the data must be on a separate volume that the ICF catalog for the logs.

Data sets that RESTORE SYSTEM uses

Table 97 lists the data sets that RESTORE SYSTEM uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set.

Table 97. Data sets that RESTORE SYSTEM uses

Data set	Description	Required?
SYSIN	An input data set that contains the utility control statement	Yes
SYSPRINT	An output data set for messages	Yes

Instructions for specific tasks

This section contains information about the following tasks:

"Restoring data in a data sharing environment"
"Using DISPLAY UTILITY with RESTORE SYSTEM" on page 548

Restoring data in a data sharing environment

Ensure that all data sharing members that were active at the SYSPITR log truncation point (or restarted after this point) have been restarted with the same

SYSPITR LRSN value. You can stop the other members of the data group (with MODE(QUIESCE)) after the SYSPITR restart.

Using DISPLAY UTILITY with RESTORE SYSTEM

To use the DISPLAY UTILITY command for RESTORE SYSTEM on a data sharing group, you must issue the command from the member on which the RESTORE SYSTEM utility is invoked.

Terminating and restarting RESTORE SYSTEM

You cannot terminate RESTORE SYSTEM by using the TERM UTILITY command.

You can restart RESTORE SYSTEM at the beginning of a phase or at the current system checkpoint. A current system checkpoint occurs during the LOGAPPLY phase after log records are processed. By default, RESTORE SYSTEM restarts at the current system checkpoint.

When you restart RESTORE SYSTEM for a data sharing group, the member on which the restart is issued must be the same member on which the original RESTORE SYSTEM was issued.

For guidance in restarting online utilities, see “Restarting an online utility” on page 43.

Concurrency and compatibility for RESTORE SYSTEM

While RESTORE SYSTEM is running, no other utilities can run.

After running RESTORE SYSTEM

Complete the following steps after running RESTORE SYSTEM:

1. Stop and start each DB2 subsystem or member to remove it from access maintenance mode.
2. Use the DISPLAY UTIL command to see if any utilities are running. If other utilities are running, use the TERM UTIL command to end them.
3. Use the RECOVER utility to recover all objects in RECOVER-pending (RECP) or REBUILD-pending (RBDP) status, or use the REBUILD INDEX utility to rebuild objects. If a CREATE TABLESPACE, CREATE INDEX, or data set extension has failed, you can also recover or rebuild any objects in the logical page list (LPL).

Sample RESTORE SYSTEM control statements

RESTORE SYSTEM uses data that is copied by the BACKUP SYSTEM utility. For a complete list of all of the steps for system-level point-in-time recovery, see Part 4 of *DB2 Administration Guide*.

Example 1: Recovering a backup system. The following control statement specifies that the RESTORE SYSTEM utility is to recover a DB2 subsystem or a data sharing group to a previous point in time by restoring volume copies and applying any outstanding log changes.

```
//STEP1    EXEC DSNUPROC,TIME=1440,
//         UTPROC='',
//         SYSTEM='DSN'
//SYSIN    DD *
  RESTORE SYSTEM
/*
```

Example 2: Recovering a backup system after the database volumes have already been restored. The LOGONLY keyword in the following control statement indicates that RESTORE SYSTEM is to apply any outstanding log changes to the database. The utility is not to restore the volume copies. In this example, the database volumes have already been restored outside of DB2. Note that RESTORE SYSTEM applies log changes; it never restores the log copy pool.

```
//STEP1    EXEC DSNUPROC,TIME=1440,
//         UTPROC='',
//         SYSTEM='DSN'
//SYSIN    DD *
  RESTORE SYSTEM LOGONLY
/*
```

Chapter 29. RUNSTATS

The RUNSTATS utility gathers summary information about the characteristics of data in table spaces, indexes, and partitions. DB2 records these statistics in the DB2 catalog and uses them to select access paths to data during the bind process. You can use these statistics to evaluate the database design and determine when table spaces or indexes must be reorganized. To obtain the updated statistics, you can query the catalog tables.

The two formats for the RUNSTATS utility are RUNSTATS TABLESPACE and RUNSTATS INDEX. RUNSTATS TABLESPACE gathers statistics on a table space and, optionally, on tables, indexes or columns; RUNSTATS INDEX gathers statistics only on indexes.

RUNSTATS can collect statistics on any single column or set of columns. RUNSTATS collects the following two types of distribution statistics:

Frequency
The percentage of rows in the table that contain a value for a column or combination of values for a set of columns.

Cardinality
The number of distinct values in the column or set of columns.

When you run RUNSTATS TABLESPACE, you can use the COLGROUP option to collect frequency and cardinality statistics on any column group. You can also collect frequency and cardinality statistics on any single column. When you run RUNSTATS INDEX, you can collect frequency statistics on the leading column of an index and multi-column frequency and cardinality statistics on the leading concatenated columns of an index.

For a diagram of RUNSTATS syntax and a description of available options, see "Syntax and options of the RUNSTATS control statement" on page 552. For detailed guidance on running this utility, see "Instructions for running RUNSTATS" on page 564.

Output: RUNSTATS updates the DB2 catalog with table space or index space statistics, prints a report, or both. See "Reviewing RUNSTATS output" on page 572 for a list of all the catalog tables and columns that are updated by RUNSTATS.

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:
- STATS privilege for the database
- DBADM, DBCTRL, or DBMAINT authority for the database
- SYSCTRL or SYSADM authority

An ID with installation SYSOPR authority can also execute the RUNSTATS utility, but only on a table space in the DSNDB06 database.

To run RUNSTATS TABLESPACE TABLE REPORT YES and see column values in the report, you must use a privilege set that includes the SELECT privilege on the catalog tables. RUNSTATS does not report values from tables that the user is not authorized to see.

To gather statistics on a LOB table space, you must have SYSADM or DBADM authority for the LOB table space.

Execution phases of RUNSTATS: The RUNSTATS utility operates in the following phases:

Phase	Description
UTILINIT	Performs initialization
RUNSTATS	Scans table space or index and updates catalog. If you specify COLGROUP, RUNSTATS also performs a subtask that sorts one or more column group's data. If you specify FREQVAL with COLGROUP or are collecting frequency statistics for data-partitioned secondary indexes, RUNSTATS also performs a subtask that sorts the partition-level frequency data.
UTILTERM	Performs cleanup

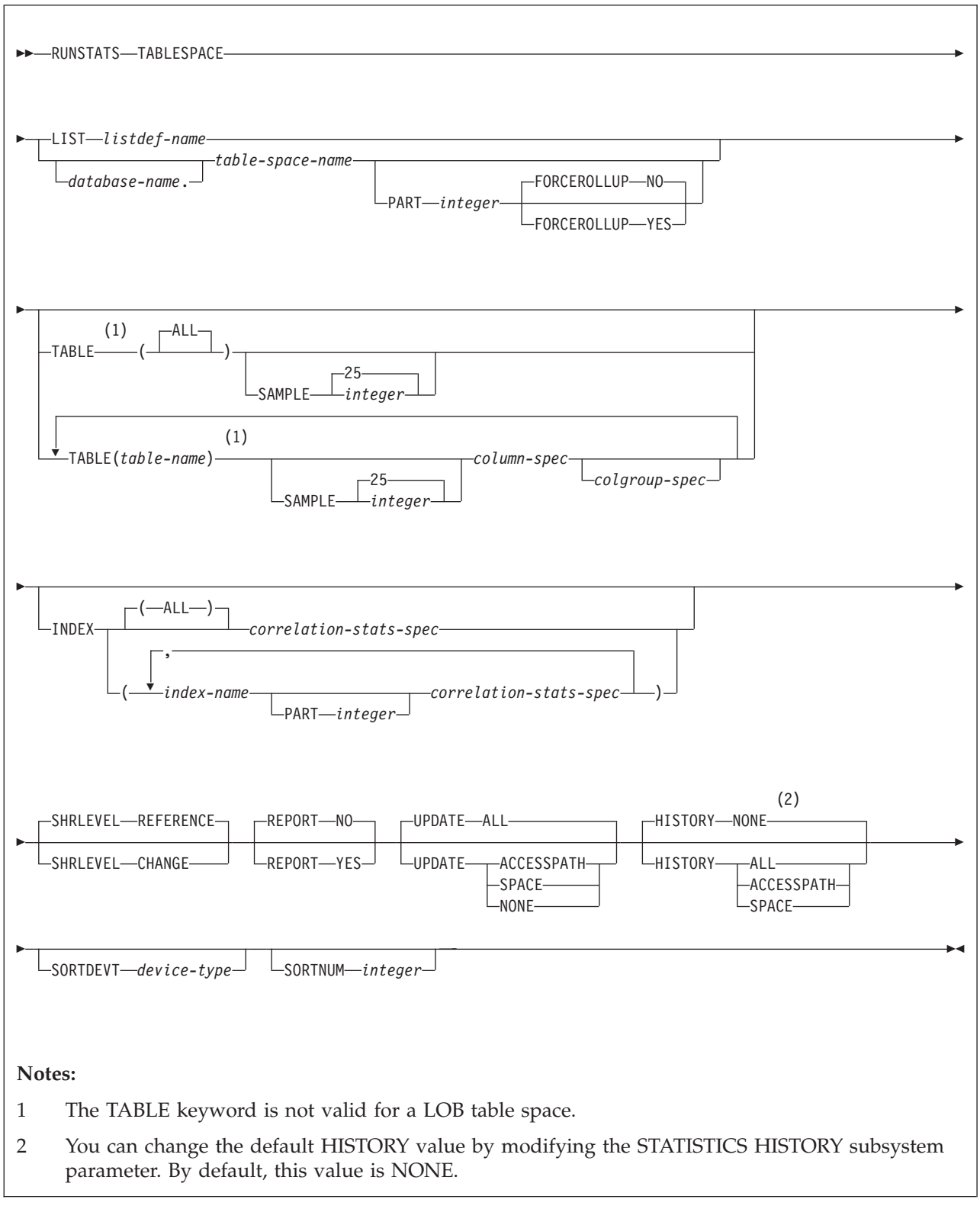
The following topics provide additional information:

- "Syntax and options of the RUNSTATS control statement"
- "Instructions for running RUNSTATS" on page 564
- "Concurrency and compatibility for RUNSTATS" on page 570
- "Reviewing RUNSTATS output" on page 572
- "After running RUNSTATS" on page 582
- "Sample RUNSTATS control statements" on page 582

Syntax and options of the RUNSTATS control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

RUNSTATS TABLESPACE syntax diagram

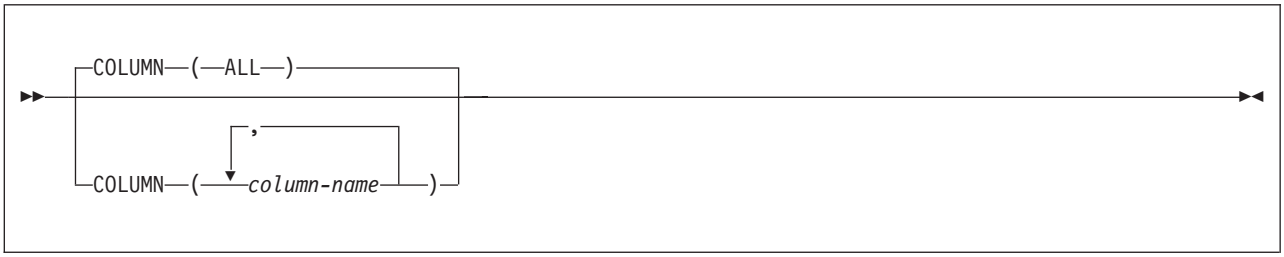


Notes:

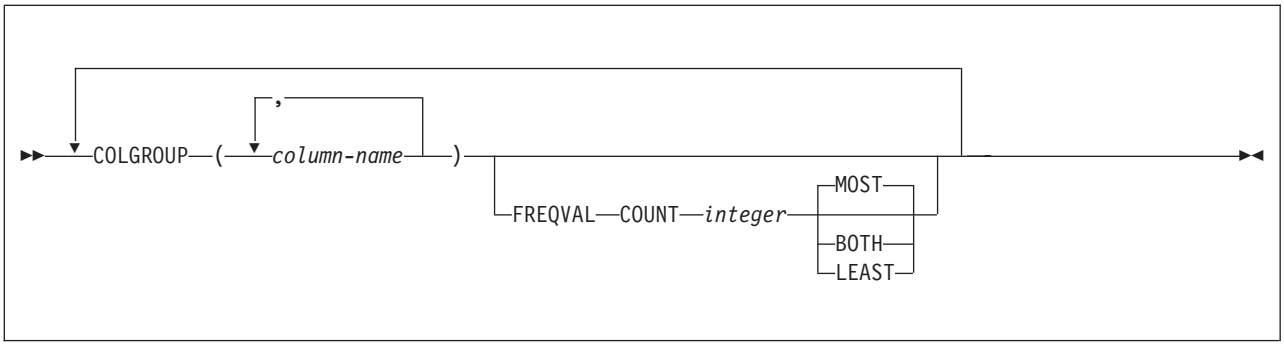
1 The TABLE keyword is not valid for a LOB table space.

2 You can change the default HISTORY value by modifying the STATISTICS HISTORY subsystem parameter. By default, this value is NONE.

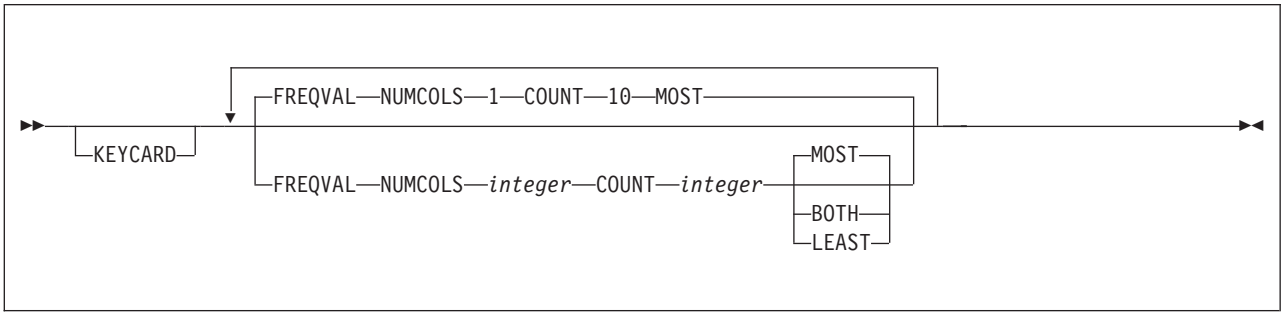
column-spec:



colgroup-spec:



correlation-stats-spec:



RUNSTATS TABLESPACE option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

TABLESPACE *database-name.table-space-name*
Specifies the table space (and, optionally, the database to which it belongs) on which table space and table statistics are to be gathered. This keyword must not identify a table space in DSNDB01 or DSNDB07.

LIST *listdef-name*
Specifies the name of a previously defined LISTDEF list name. You can specify one LIST keyword for each RUNSTATS control statement. When you specify this keyword with RUNSTATS TABLESPACE, the list must contain only table spaces. Do not specify LIST with keywords from the TABLE...(*table-name*) specification. Instead, specify LIST with TABLE (ALL). Likewise, do not specify LIST with keywords from the

INDEX...(*index-name*) specification. You cannot specify index names with a list. Use INDEX(ALL) instead.

If you specify LIST, you cannot specify the PART option. Instead, use the PARTLEVEL option on the LISTDEF statement. The TABLESPACE keyword is required in order to validate the contents of the list. RUNSTATS TABLESPACE is invoked once for each item in the list.

For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

database-name
Identifies the name of the database to which the table space belongs. The **default** is **DSNDB04**.

table-space-name
Identifies the name of the table space on which statistics are to be gathered.

If the table space that is specified by the TABLESPACE keyword is a LOB table space, you can specify only the following additional keywords: SHRLEVEL REFERENCE or CHANGE, REPORT YES or NO, and UPDATE ALL or NONE.

PART *integer*
Identifies a table space partition on which statistics are to be collected.

integer is the number of the partition and must be in the range from 1 to the number of partitions that are defined for the table space. The maximum is 4096.

You cannot specify PART with LIST.

TABLE
Specifies the table on which column statistics are to be gathered. All tables must belong to the table space that is specified in the TABLESPACE option. You cannot specify the TABLE option for a LOB table space.

(ALL) Specifies that column statistics are to be gathered on all columns of all tables in the table space. The **default** is **ALL**.

(*table-name***)**
Specifies the tables on which column statistics are to be gathered. If you omit the qualifier, RUNSTATS uses the user identifier for the utility job as the qualifier. Enclose the table name in quotation marks if the name contains a blank.

If you specify more than one table, you must repeat the TABLE option. Multiple TABLE options must be specified entirely before or after any INDEX keyword that may also be specified. For example, the INDEX keyword may not be specified between any two TABLE keywords.

SAMPLE *integer*
Indicates the percentage of rows that RUNSTATS is to sample when collecting statistics on non-leading-indexed columns. You can specify any value from 1 through 100. The **default** is **25**.

You cannot specify SAMPLE for LOB table spaces.

COLUMN
Specifies columns on which column statistics are to be gathered.

You can specify this option only if you specify a particular table on which statistics are to be gathered. (Use the TABLE (*table-name*) option to specify a

particular table.) If you specify particular tables and do not specify the COLUMN option, RUNSTATS uses the default, COLUMN(ALL). If you do not specify a particular table when using the TABLE option, you cannot specify the COLUMN option; however, in this case, COLUMN(ALL) is assumed.

(ALL)
Specifies that statistics are to be gathered on all columns in the table.

The COLUMN (ALL) option is not allowed for LOB table spaces.

(*column-name***, ...)**
Specifies the columns on which statistics are to be gathered. You can specify a list of column names. If you specify more than one column, separate each name with a comma.

The more columns that you specify, the longer the job takes to complete.

COLGROUP (*column-name***, ...)**
Indicates that the specified set of columns are to be treated as a group. This option enables RUNSTATS to collect a cardinality value on the specified column group.

When you specify the COLGROUP keyword, RUNSTATS collects correlation statistics for the specified column group. If you want RUNSTATS to also collect distribution statistics, specify the FREQVAL option with COLGROUP.

(*column-name*, ...) specifies the names of the columns that are part of the column group.

To specify more than one column group, repeat the COLGROUP option.

FREQVAL
Indicates, when specified with the COLGROUP option, that frequency statistics are also to be gathered for the specified group of columns. (COLGROUP indicates that cardinality statistics are to be gathered.) One group of statistics is gathered for each column. You must specify COUNT *integer* with COLGROUP FREQVAL.

COUNT *integer*
Indicates the number of frequently occurring values to be collected from the specified column group. For example, COUNT 20 means that DB2 collects 20 frequently occurring values from the column group. You must specify a value for *integer;* no default value is assumed.

Be careful when specifying a high value for COUNT. Specifying a value of 1000 or more can increase the prepare time for some SQL statements.

MOST
Indicates that the utility is to collect the most frequently occurring values for the specified set of columns when COLGROUP is specified. The **default** is **MOST**.

LEAST
Indicates that the utility is to collect the least frequently occurring values for the specified set of columns when COLGROUP is specified.

BOTH
Indicates that the utility is to collect the most and the least frequently occurring values for the specified set of columns when COLGROUP is specified.

INDEX
Specifies indexes on which statistics are to be gathered. RUNSTATS gathers

column statistics for the first column of the index, and possibly additional index columns depending on the options that you specify. All the indexes must be associated with the **same** table space, which must be the table space that is specified in the TABLESPACE option.

INDEX can be used on auxiliary tables to gather statistics on an index.

(ALL) Specifies that column statistics are to be gathered for all indexes that are defined on tables that are contained in the table space. The **default** is **ALL**.

(*index-name***, ...)**
Specifies the indexes for which statistics are to be gathered. You can specify a list of index names. If you specify more than one index, separate each name with a comma. Enclose the index name in quotation marks if the name contains a blank.

PART *integer*
Identifies an index partition on which statistics are to be collected.

integer is the number of the partition.

KEYCARD
Collects all of the distinct values in all of the 1 to *n* key column combinations for the specified indexes. *n* is the number of columns in the index. For example, suppose that you have an index defined on three columns: A, B, and C. If you specify KEYCARD, RUNSTATS collects cardinality statistics for column A, column set A and B, and column set A, B, and C.

FREQVAL
Controls, when specified with the INDEX option, the collection of frequent-value statistics. If you specify FREQVAL with INDEX, this keyword must be followed by the NUMCOLS and COUNT keywords.

NUMCOLS *integer*
Indicates the number of columns in the index for which RUNSTATS is to collect frequently occurring values. *integer* can be a number between 1 and the number of indexed columns. If you specify a number greater than the number of indexed columns, RUNSTATS uses the number of columns in the index.

For example, suppose that you have an index defined on three columns: A, B, and C. If you specify NUMCOLS 1, DB2 collects frequently occurring values for column A. If you specify NUMCOLS 2, DB2 collects frequently occurring values for the column set A and B. If you specify NUMCOLS 3, DB2 collects frequently occurring values for the column set A, B, and C.

The **default** is **1**, which means that RUNSTATS is to collect frequently occurring values on the first key column of the index.

COUNT*integer*
Indicates the number of frequently occurring values that are to be collected from the specified key columns. For example, specifying 15 means that RUNSTATS is to collect 15 frequently occurring values from the specified key columns. The **default** is **10**.

SHRLEVEL
Indicates whether other programs that access the table space while RUNSTATS is running must use read-only access or can change the table space.

REFERENCE
Allows only read-only access by other programs. The **default** is **REFERENCE**.

CHANGE
Allows other programs to change the table space or index. With SHRLEVEL CHANGE, RUNSTATS might collect statistics on uncommitted data.

REPORT
Specifies whether RUNSTATS is to generate a set of messages that report the collected statistics.

NO
Indicates that RUNSTATS is not to generate the set of messages. The **default** is **NO**.

YES
Indicates that the set of messages is to be sent as output to SYSPRINT. The messages that RUNSTATS generates are dependent on the combination of keywords in the utility control statement. However, these messages are **not** dependent on the value of the UPDATE option. REPORT YES always generates a report of space and access path statistics.

UPDATE
Indicates which collected statistics are to be inserted into the catalog tables.

ALL Indicates that all collected statistics are to be updated in the catalog. The **default** is **ALL**.

ACCESSPATH
Indicates that DB2 is to update the catalog with only those statistics that are used for access path selection.

SPACE
Indicates that DB2 is to update the catalog with only space-related statistics.

NONE
Indicates that no catalog tables are to be updated with the collected statistics.

Executing RUNSTATS always invalidates the dynamic cache; however, when you specify UPDATE NONE REPORT NO, RUNSTATS invalidates statements in the dynamic statement cache without collecting statistics, updating catalogs tables, or generating reports.

HISTORY
Indicates which statistics are to be recorded in the catalog history tables. The value that you specify for HISTORY does not depend on the value that you specify for UPDATE.

The default is the value of the STATISTICS HISTORY subsystem parameter on the DSNTIPO installation panel. By default, this parameter value is NONE.

ALL Indicates that all collected statistics are to be updated in the catalog history tables.

ACCESSPATH
Indicates that DB2 is to update the catalog history tables with only those statistics that are used for access path selection.

SPACE
Indicates that DB2 is to update the catalog history tables with only space-related statistics.

NONE
Indicates that no catalog history tables are to be updated with the collected statistics.

SORTDEVT
Specifies the device type that DFSORT uses to dynamically allocate the sort work data sets that are required.

device-type
Specifies any device type that is acceptable for the DYNALLOC parameter of the SORT or OPTIONS option of DFSORT. For information about valid device types, see *DFSORT Application Programming Guide*.

If you omit SORTDEVT, a sort is required, and you have not provided the DD statements that the SORT program requires for the temporary data sets, SORTDEVT will default to SYSALLDA and the temporary data sets will be dynamically allocated.

If you specify SORTDEVT and omit SORTNUM, no value is passed to DFSORT; DFSORT uses its own default.

SORTNUM *integer*
Specifies the number of required sort work data sets that DFSORT is to allocate.

integer is the number of temporary data sets that can range from 2 to 255.

You need at least two sort work data sets for each sort. The SORTNUM value applies to each sort invocation in the utility. For example, if there are three indexes, SORTKEYS is specified, there are no constraints limiting parallelism, and SORTNUM is specified as 8, then a total of 24 sort work data sets will be allocated for a job.

Each sort work data set consumes both above the line and below the link virtual storage, so if you specify too high a value for SORTNUM, the utility may decrease the degree of parallelism due to virtual storage constraints, and possibly decreasing the degree down to one, meaning no parallelism.

Important: The SORTNUM keyword will not be considered if ZPARM UTSORTAL is set to YES and IGNSORTN is set to YES.

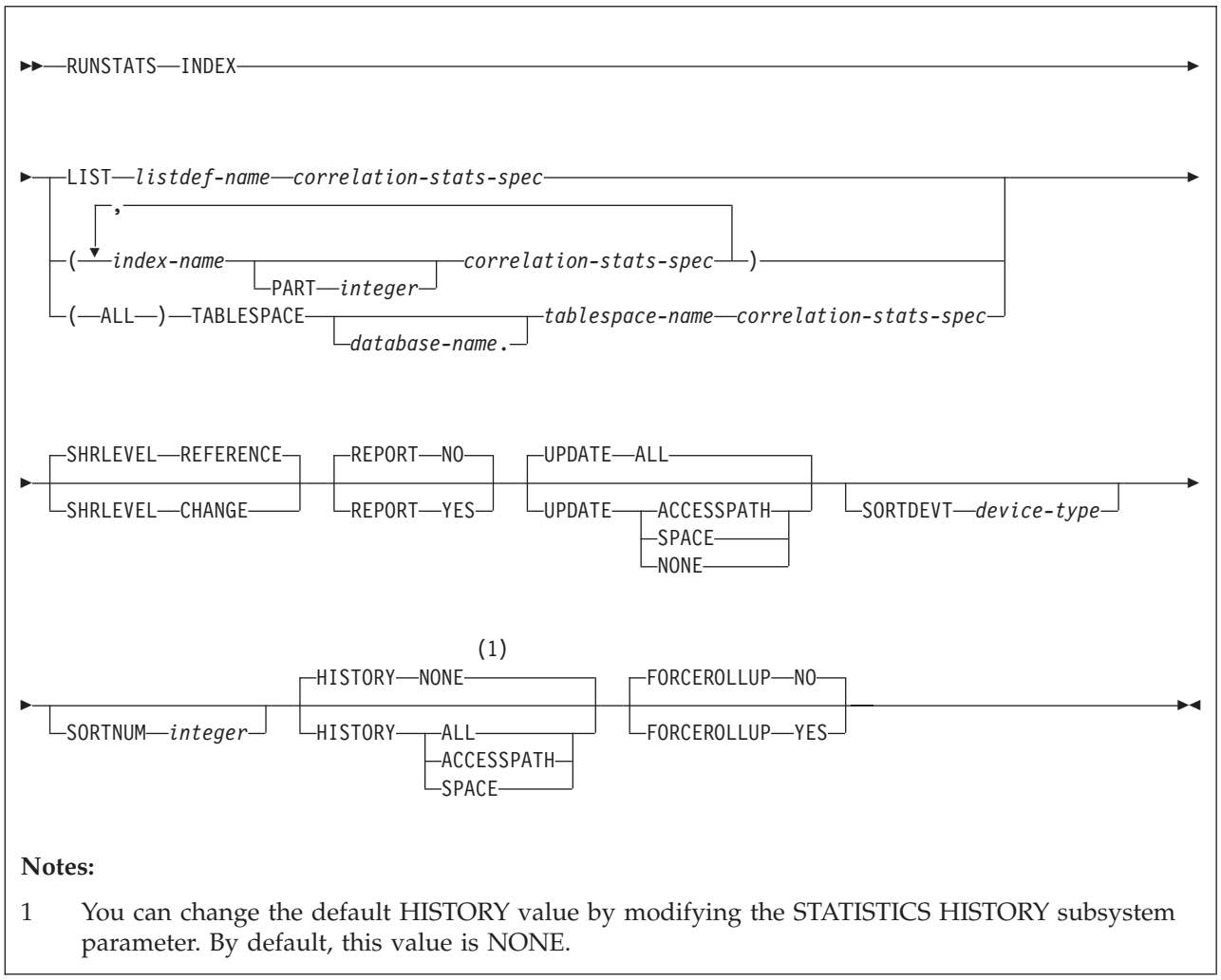
FORCEROLLUP
Specifies whether aggregation or rollup of statistics is to occur even if statistics have not been gathered on some partitions. This option enables the optimizer to select the best access path.

YES Indicates that forced aggregation or rollup processing is to be done, even though some partitions might not contain data.

NO Indicates that aggregation or rollup is to be done only if data is available for all partitions.

If the value for STATISTICS ROLLUP on panel DSNTIPO is NO and data is not available for all partitions, DB2 issues message DSNU623I.

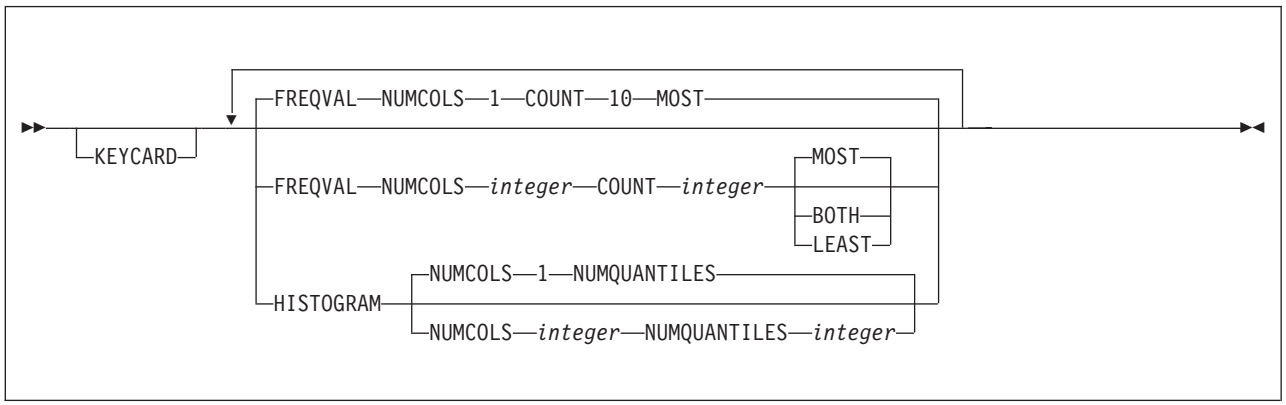
RUNSTATS INDEX syntax diagram



Notes:

1 You can change the default HISTORY value by modifying the STATISTICS HISTORY subsystem parameter. By default, this value is NONE.

correlation-stats-spec:



RUNSTATS INDEX option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

INDEX
Specifies the indexes on which statistics are to be gathered. Column statistics are gathered on the first column of the index. All of the indexes must be associated with the **same** table space.

LIST *listdef-name*
Specifies the name of a previously defined LISTDEF list name. You can specify one LIST keyword for each RUNSTATS control statement. When you specify LIST with RUNSTATS INDEX, the list must contain only index spaces. Do not specify LIST with keywords from the INDEX...(*index-name*) specification; except for the correlation-stats-spec.

RUNSTATS groups indexes by their related table space. RUNSTATS INDEX is invoked once per table space. The INDEX keyword is required in order to validate the contents of the LIST.

For more information about LISTDEF specifications, see Chapter 15, "LISTDEF," on page 173.

(*index-name***, ...)**
Specifies the indexes on which statistics are to be gathered. You can specify a list of index names. If you specify more than one index, separate each name with a comma. Enclose the index name in quotation marks if the name contains a blank.

PART *integer*
Identifies the index partition on which statistics are to be collected.

integer is the number of the partition.

(ALL)
Specifies that statistics are to be gathered on all indexes that are defined on all tables in the specified table space.

TABLESPACE
Identifies the table space and, optionally, the database to which it belongs, for which index statistics are to be gathered.

database-name
The name of the database to which the table space belongs. The **default** is **DSNDB04**.

tablespace-name
The name of the table space for which index statistics are to be gathered.

KEYCARD
Collects all of the distinct values in all of the 1 to *n* intermediate key column
combinations for the specified indexes. *n* is the number of columns in the
index. For example, suppose that you have an index defined on three columns:
A, B, and C. If you specify KEYCARD, RUNSTATS collects cardinality statistics
for the intermediate column set A and B.

FREQVAL
Controls, when specified with the INDEX option, the collection of frequent-value statistics. If you specify FREQVAL with INDEX, this keyword must be followed by the NUMCOLS and COUNT keywords.

| **NUMCOLS** *integer*
| Indicates the number of columns in the index for which RUNSTATS is to
| collect frequently occurring values. *integer* can be a number between 1 and

the number of indexed columns. If you specify a number greater than the number of indexed columns, RUNSTATS uses the number of columns in the index.

For example, suppose that you have an index defined on three columns: A, B, and C. If you specify NUMCOLS 1, DB2 collects frequently occurring values for column A. If you specify NUMCOLS 2, DB2 collects frequently occurring values for the column set A and B. If you specify NUMCOLS 3, DB2 collects frequently occurring values for the column set A, B, and C.

The **default** is **1**, which means that RUNSTATS is to collect frequently occurring values on the first key column of the index.

COUNT *integer*
Indicates the number of frequently occurring values that are to be collected from the specified key columns. For example, specifying 15 means that RUNSTATS is to collect 15 frequently occurring values from the specified key columns. The **default** is **10**.

MOST
Indicates that the utility is to collect the most frequently occurring values for the numcols specified for the index. The **default** is **MOST**.

LEAST
Indicates that the utility is to collect the least frequently occurring values for the numcols specified for the index.

BOTH
Indicates that the utility is to collect the most and the least frequently occurring values for the numcols specified for the index.

SHRLEVEL
Indicates whether other programs that access the table space while RUNSTATS is running must use read-only access or can change the table space.

REFERENCE
Allows only read-only access by other programs. The **default** is **REFERENCE**.

CHANGE
Allows other programs to change the table space or index. With SHRLEVEL CHANGE, RUNSTATS might collect statistics on uncommitted data.

REPORT
Specifies whether RUNSTATS is to generate a set of messages that report the collected statistics.

NO
Indicates that RUNSTATS is not to generate the set of messages. The **default** is **NO**.

YES
Indicates that the set of messages is to be sent as output to SYSPRINT. The messages that RUNSTATS generates are dependent on the combination of keywords in the utility control statement. However, these messages are **not** dependent on the value of the UPDATE option. REPORT YES always generates a report of space and access path statistics.

UPDATE
Indicates which collected statistics are to be inserted into the catalog tables.

ALL Indicates that all collected statistics are to be updated in the catalog. The **default** is **ALL**.

ACCESSPATH
Indicates that DB2 is to update the catalog with only those statistics that are used for access path selection.

SPACE
Indicates that DB2 is to update the catalog with only space-related statistics.

NONE
Indicates that no catalog tables are to be updated with the collected statistics.

Executing RUNSTATS always invalidates the dynamic cache; however, when you specify UPDATE NONE REPORT NO, RUNSTATS invalidates statements in the dynamic statement cache without collecting statistics, updating catalogs tables, or generating reports.

SORTDEVT
Specifies the device type that DFSORT uses to dynamically allocate the sort work data sets that are required.

device-type
Specifies any device type that is acceptable for the DYNALLOC parameter of the SORT or OPTIONS option of DFSORT. For information about valid device types, see *DFSORT Application Programming Guide*.

If you omit SORTDEVT, a sort is required, and you have not provided the DD statements that the SORT program requires for the temporary data sets, SORTDEVT will default to SYSALLDA and the temporary data sets will be dynamically allocated.

If you specify SORTDEVT and omit SORTNUM, no value is passed to DFSORT; DFSORT uses its own default.

SORTNUM
Specifies the number of required sort work data sets that DFSORT is to allocate.

integer is the number of temporary data sets that can range from 2 to 255.

You need at least two sort work data sets for each sort. The SORTNUM value applies to each sort invocation in the utility. For example, if there are three indexes, SORTKEYS is specified, there are no constraints limiting parallelism, and SORTNUM is specified as 8, then a total of 24 sort work data sets will be allocated for a job.

Each sort work data set consumes both above the line and below the link virtual storage, so if you specify too high a value for SORTNUM, the utility may decrease the degree of parallelism due to virtual storage constraints, and possibly decreasing the degree down to one, meaning no parallelism.

HISTORY
Indicates which statistics are to be recorded in the catalog history tables. The value that you specify for HISTORY does not depend on the value that you specify for UPDATE.

The default is the value of the STATISTICS HISTORY subsystem parameter on the DSNTIPO installation panel. By default, this parameter value is NONE.

ALL Indicates that all collected statistics are to be updated in the catalog history tables.

ACCESSPATH
Indicates that DB2 is to update the catalog history tables with only those statistics that are used for access path selection.

SPACE
Indicates that DB2 is to update the catalog history tables with only space-related statistics.

NONE
Indicates that no catalog history tables are to be updated with the collected statistics.

FORCEROLLUP
Specifies whether aggregation or rollup of statistics is to occur even if statistics have not been gathered on some partitions. This option enables the optimizer to select the best access path.

YES Indicates that forced aggregation or rollup processing is to be done, even though some partitions might not contain data.

NO Indicates that aggregation or rollup is to be done only if data is available for all partitions.

If the value for STATISTICS ROLLUP on panel DSNTIPO is NO and data is not available for all partitions, DB2 issues message DSNU623I.

Instructions for running RUNSTATS

To run RUNSTATS, you must:

1. Read "Before running RUNSTATS" in this section.
2. Prepare the necessary data sets, as described in "Data sets that RUNSTATS uses" on page 565.
3. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.
4. Prepare a utility control statement, that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks" on page 567.
5. Check the compatibility table in "Concurrency and compatibility for RUNSTATS" on page 570 if you want to run other jobs concurrently on the same target objects.
6. Plan for restart if the RUNSTATS job doesn't complete, as described in "Terminating or restarting RUNSTATS" on page 570. RUNSTATS can be restarted, but it starts over again from the beginning.
7. Run RUNSTATS by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Before running RUNSTATS

You can use SQL to manually update the catalog columns that RUNSTATS updates. Use caution when running RUNSTATS after any user has manually updated the statistic columns in the catalog. RUNSTATS replaces any values that the user changed.

Restriction: RUNSTATS might not provide useful statistics on encrypted data.

Data sets that RUNSTATS uses

Table 98 lists the data sets that RUNSTATS uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 98. Data sets that RUNSTATS uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
RNPRIN01	A data set that contains messages from DFSORT (usually, SYSOUT or DUMMY). This data set is used when distribution statistics are collected for column groups.	No[1]
STPRIN01	A data set that contains messages from DFSORT (usually, SYSOUT or DUMMY). This data set is used when frequency statistics are collected on DPSI's or when TABLESPACE TABLE COLGROUP FREQVAL is specified	No[1, 2]
Sort work data sets	Temporary data sets for sort input and output when collecting statistics on at least one data-partitioned secondary index. The DD names have the form ST01WK*nn* or ST02WK*nn*. ST02WK*nn* will also be used with STATWK01 for RUNSTATS with COLGROUP and FREQVAL option.	No[2, 3, 5]
Sort work data sets	Temporary data sets for sort input and output when collecting distribution statistics for column groups. The DD names have the form STATWK01.	No[1, 3, 5]

Notes:

1. Required when collecting distribution statistics for column groups.
2. Required when collecting statistics on at least one data-partitioned secondary index or when TABLESPACE TABLE COLGROUP FREQVAL is specified.
3. If the DYNALLOC parm of the SORT program is not turned on, you need to allocate the data set. Otherwise, DFSORT dynamically allocates the temporary data set.
4. Required when the COLGROUP with FREQVAL options are specified.
5. It is recommended that you use dynamic allocation by specifying SORTDEVT in the utility statement because dynamic allocation reduces the maintenance required of the utility job JCL.

The following objects are named in the utility control statement and do not require DD statements in the JCL:

Table space or index
Object that is to be scanned.

Calculating the size of the sort work data sets: Depending on the type of statistics that RUNSTATS collects, the utility uses the ST01WK*nn* data sets, the SORTWK01 data set, both types of data sets, or neither. RUNSTATS with COLGROUP and FREQVAL option will use both SORTWK01 and ST01WK*nn* data sets.

The ST01WK*nn* data sets are used when collecting statistics on at least one data-partitioned secondary index. To calculate the approximate size (in bytes) of the ST01WK*nn* data set, use the following formula:

2 ×(*maximum record length* × *numcols* × (*count* + 2) × number of indexes)

The variables in the preceding formula have the following values:

maximum record length
Maximum record length of the SYSCOLDISTSTATS record that is processed when collecting frequency statistics (You can obtain this value from the RECLENGTH column in SYSTABLES.)

numcols
Number of key columns to concatenate when you collect frequent values from the specified index.

count Number of frequent values that RUNSTATS is to collect.

The SORTWK01 data set is used when collecting distribution statistics. To calculate the approximate size (in bytes) of the SORTWK01 data set, use the following formula:

(longest_record_length + prefix) × sum from 1 to *N* ($\#colgroups_n$ × #rows - n)

The variables in the preceding formula have the following values:

N Number of tables for which distribution statistics are collected

$\#colgroups_n$
Number of column groups that are specified for the nth table

#rows Number of rows for the nth table

The ST02WK*nn* data sets are used when collecting frequency statistics on at least one COLGROUP. To calculate the approximate size (in bytes) of the ST02WK*nn* data set, use the following formula:

2 ×(*maximum record length* ×(*count* + 2) × number of indexes)

The variables in the preceding formula have the following values:

maximum record length
Maximum record length of the SYSCOLDISTSTATS record that is processed when collecting frequency statistics (You can obtain this value from the RECLENGTH column in SYSTABLES.)

count Number of frequent values that RUNSTATS is to collect.

DB2 utilities uses DFSORT to perform sorts. Sort work data sets cannot span volumes. Smaller volumes require more sort work data sets to sort the same amount of data; therefore, large volume sizes can reduce the number of needed sort work data sets. It is recommended that at least 1.2 times the amount of data to be sorted be provided in sort work data sets on disk. For more information about DFSORT, see *DFSORT Application Programming Guide*.

Creating the control statement

Create the utility control statement for the RUNSTATS job. See "Syntax and options of the RUNSTATS control statement" on page 552 for RUNSTATS syntax and

option descriptions. See "Sample RUNSTATS control statements" on page 582 for examples of RUNSTATS control statements.

Instructions for specific tasks

This section includes information about the following tasks:

Deciding when to use RUNSTATS

DB2 uses the statistics that RUNSTATS generates to determine access paths to data. If no statistics are available, DB2 makes fixed default assumptions. To ensure the effectiveness of the paths selected, run RUNSTATS at the following times:

- After a table is loaded
- After an index is physically created
- After a table space is reorganized if inline statistics were not collected
- After running extensive updates, deletions, or insertions in a table space
- After running any of the following utilities without collecting inline statistics: RECOVER TABLESPACE, REBUILD INDEX, or REORG INDEX
- Before running REORG with the OFFPOSLIMIT, INDREFLIMIT, or LEAFDISTLIMIT options
- After running the ALTER TABLE ROTATE PARTITION statement run RUNSTATS with REORG .

You should recollect frequency statistics when either of the following situations is true:

- The distribution of the data changes
- The values over which the data is distributed change

Determining when to gather statistics and what statistics to gather depends on a number of factors. The preceding information is only a guideline. You should determine your own statistic collection strategy. For more detailed information about how to determine which statistics you need and how to keep them current, see Part 5 (Volume 2) of *DB2 Administration Guide*.

One common situation in which old statistics can affect query performance is when a table has columns that contain data or ranges that are constantly changing (for example, dates and timestamps). These types of columns can result in old values in the HIGH2KEY and LOW2KEY columns in the catalog. You should periodically collect column statistics on these changing columns so that the values in HIGH2KEY and LOW2KEY accurately reflect the true range of data, and range predicates can obtain accurate filter factors.

Assessing table space status

Changes to a table space can also change its space requirements and performance. You can use RUNSTATS to update the table space statistics and then assess the current status of the table space and decide whether to reorganize or redesign the table space.

Collecting distribution statistics for column groups

When RUNSTATS collects distribution statistics for columns groups, the utility invokes DFSORT or a similar product to sort the distribution statistics. This sort requires its own work data set. You can let this data set be dynamically allocated through the SORT program, or you can allocate the data set through a DD statement in the job JCL. The DD name is STATWK01.

If you need to control the size or placement of the data sets, use the JCL statements to allocate STATWK01. To estimate the size of this sort work data set, use the formula for STATWK01 in "Data sets that RUNSTATS uses" on page 565.

To let the work data set be dynamically allocated, remove the STATWK01 DD statements from the job and allocate the UTPRINT statement to SYSOUT. If you let the SORT program dynamically allocate this data set, you must specify the SORTDEV option in the RUNSTATS control statement.

Updating statistics for a partitioned table space

You can run RUNSTATS on one or more single partitions of one or more table spaces or indexes (including data-partitioned secondary indexes). When you run the utility on a single partition of an object, RUNSTATS uses the resulting partition-level statistics to update the aggregate statistics for the entire object.

Running RUNSTATS on the DB2 catalog

You can run RUNSTATS on the DB2 catalog to gather index space and table space statistics for various catalog tables. The sample in Figure 103 shows part of the output from a RUNSTATS job on a catalog table space and its indexes:

```
DSNU000I   DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = DSNTEX
DSNU050I   DSNUGUTC - RUNSTATS TABLESPACE DSNDB06.SYSDBASE INDEX(ALL)
DSNU610I # DSNUSUTP - SYSTABLEPART CATALOG UPDATE FOR DSNDB06.SYSDBASE SUCCESSFUL
DSNU610I # DSNUSUTS - SYSTABLESPACE CATALOG UPDATE FOR DSNDB06.SYSDBASE SUCCESSFUL
DSNU610I # DSNUSUTB - SYSTABLES CATALOG UPDATE FOR SYSIBM.SYSTABLESPACE SUCCESSFUL

DSNU610I # DSNUSUTB - SYSTABLES CATALOG UPDATE FOR SYSIBM.SYSSYNONYMS SUCCESSFUL
DSNU610I # DSNUSUIX - SYSINDEXES CATALOG UPDATE FOR SYSIBM.DSNDSX01 SUCCESSFUL
DSNU610I # DSNUSUIP - SYSINDEXPART CATALOG UPDATE FOR SYSIBM.DSNDSX01 SUCCESSFUL
DSNU610I # DSNUSUCO - SYSCOLUMNS CATALOG UPDATE FOR SYSIBM.DSNDSX01 SUCCESSFUL
DSNU610I # DSNUSUFL - SYSFIELDS CATALOG UPDATE FOR SYSIBM.DSNDSX01 SUCCESSFUL

DSNU610I # DSNUSUIX - SYSINDEXES CATALOG UPDATE FOR SYSIBM.DSNDYX01 SUCCESSFUL
DSNU610I # DSNUSUIP - SYSINDEXPART CATALOG UPDATE FOR SYSIBM.DSNDYX01 SUCCESSFUL
DSNU610I # DSNUSUCO - SYSCOLUMN CATALOG UPDATE FOR SYSIBM.DSNDYX01 SUCCESSFUL
DSNU610I # DSNUSUFL - SYSFIELDS CATALOG UPDATE FOR SYSIBM.DSNDYX01 SUCCESSFUL
DSNU010I   DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 103. Example RUNSTATS output from a job on a catalog table space

DB2 uses the collected statistics on the catalog to determine the access path for user queries of the catalog.

Improving performance

You can improve the performance of RUNSTATS on table spaces that are defined with the LARGE option by specifying the SAMPLE option, which reduces the number of rows that are scanned for statistics.

Consider running several RUNSTATS jobs concurrently against different partitions of a partitioned table space or index rather than running a single RUNSTATS job on the entire table space or index. The sum of the processor time for the concurrent jobs is roughly equivalent to the processor time for running the single

RUNSTATS job. However, the total elapsed time for the concurrent jobs can be significantly less than when you run RUNSTATS on an entire table space or index.

Run RUNSTATS on only the columns or column groups that might be used as search conditions in a WHERE clause of queries. Use the COLGROUP option to identify the column groups. Collecting additional statistics on groups of columns that are used as predicates improves the accuracy of the filter factor estimate and leads to improved query performance. Collecting statistics on all columns of a table is costly and might not be necessary.

In some cases, you can avoid running RUNSTATS by specifying the STATISTICS keyword in LOAD, REBUILD INDEX, or REORG utility statements. When you specify STATISTICS in one of these utility statements, DB2 updates the catalog with table space or index space statistics for the objects on which the utility is run. However, you cannot collect column group statistics with the STATISTICS keyword. You can collect column group statistics only by running the RUNSTATS utility. If you restart a LOAD or REBUILD INDEX job that uses the STATISTICS keyword, DB2 does not collect inline statistics. For these cases, you need to run the RUNSTATS utility after the restarted utility job completes. For information about restarting a REORG job that uses the STATISTICS keyword, see"Restarting REORG STATISTICS" on page 479.

Collecting frequency statistics for data-partitioned secondary indexes

When RUNSTATS collects frequency statistics on at least one data-partitioned secondary index, the utility invokes DFSORT or a similar product to sort the statistics. This sort requires temporary sort work data sets. You can let these data sets be dynamically allocated through the SORT program, or you can allocate the data sets through DD statements in the job JCL. The DD name is ST01WK*nn*.

If you need to control the size or placement of the data sets, use the JCL statements to allocate ST01WK*nn*. To estimate the size of this sort work data set, use the formula for ST01WK*nn* in "Data sets that RUNSTATS uses" on page 565.

To let the sort work data sets be dynamically allocated, remove the ST01WK*nn* DD statements from the job and allocate the UTPRINT statement to SYSOUT. If you let the SORT program dynamically allocate these data sets, you must specify the SORTDEV option in the RUNSTATS control statement to specify the device type for the temporary data sets. Optionally, you can also use the SORTNUM option to specify the number of temporary data sets to use.

Note: Runstats for the cardinality statistics collection for DPSI SYSIBM.SYSINDEXES&SYSIBM.SYSTABLES is estimated.

Invalidating statements in the dynamic statement cache

DB2 invalidates statements in the dynamic statement cache when you run RUNSTATS on objects to which those statements refer. In a data sharing environment, the relevant statements are also invalidated in the cache of other members in the group. DB2 invalidates the cached statements to ensure that the next invocations of those statements are fully prepared and that they use the latest access path changes. You can invalidate statements in the dynamic statement cache without collecting statistics by specifying the options UPDATE NONE and REPORT NO.

Collecting statistics history

Use the HISTORY option to collect the statistics history. When you specify HISTORY with a value other than NONE, RUNSTATS updates the catalog history tables with the access path statistics, space statistics, or both, depending on the parameter that you specify with HISTORY. The HISTORY option does not update the main catalog statistics that DB2 uses to select access paths. You can use the HISTORY option to monitor how statistics change over time without updating the main catalog statistics that DB2 uses to select access paths.

Collecting statistics on LOB table spaces

You can specify that RUNSTATS is to collect space statistics on a LOB table space so that you can determine when the LOB table space should be reorganized. No statistics on the LOB table space affect access path selection.

Terminating or restarting RUNSTATS

You can terminate RUNSTATS with the TERM UTILITY command.

You can restart a RUNSTATS utility job, but it starts from the beginning again. For guidance in restarting online utilities, see "Restarting an online utility" on page 43.

Concurrency and compatibility for RUNSTATS

DB2 treats individual data and index partitions as distinct target objects. Utilities operating on different partitions of the same table space or index space are compatible.

Table 99 shows which claim classes RUNSTATS claims and drains and any restrictive state that the utility sets on the target object.

Table 99. Claim classes of RUNSTATS operations

Target	RUNSTATS TABLESPACE SHRLEVEL REFERENCE	RUNSTATS TABLESPACE SHRLEVEL CHANGE	RUNSTATS INDEX SHRLEVEL REFERENCE	RUNSTATS INDEX SHRLEVEL CHANGE
Table space or partition	DW/UTRO	CR/UTRW[1]	None	None
Index or partition	None	None	DW/UTRO	CR/UTRW

Legend:
- DW - Drain the write claim class - concurrent access for SQL readers.
- CR - Claim the read claim class.
- UTRO - Utility restrictive state - read-only access allowed.
- UTRW - Utility restrictive state - read-write access allowed.
- None - Object is not affected by this utility.

Notes:

1. If the target object is a segmented table space, SHRLEVEL CHANGE does not allow you to concurrently execute an SQL searched DELETE without the WHERE clause.

Table 100 on page 571 shows which utilities can run concurrently with RUNSTATS on the same target object. The target object can be a table space, an index space, or a partition of a table space or index space. If compatibility depends on particular options of a utility, that information is also shown in the table.

Table 100. Compatibility of RUNSTATS with other utilities

Utility	RUNSTATS TABLESPACE SHRLEVEL REFERENCE	RUNSTATS TABLESPACE SHRLEVEL CHANGE	RUNSTATS INDEX SHRLEVEL REFERENCE	RUNSTATS INDEX SHRLEVEL CHANGE
CHECK DATA DELETE NO	Yes	Yes	Yes	Yes
CHECK DATA DELETE YES	No	No	No	No
CHECK INDEX	Yes	Yes	Yes	Yes
CHECK LOB	Yes	Yes	Yes	Yes
COPY INDEXSPACE	Yes	Yes	Yes	Yes
COPY TABLESPACE	Yes	Yes	Yes	Yes
DIAGNOSE	Yes	Yes	Yes	Yes
LOAD	No	No	No	No
MERGECOPY	Yes	Yes	Yes	Yes
MODIFY RECOVERY	Yes	Yes	Yes	Yes
QUIESCE	Yes	Yes	Yes	Yes
REBUILD INDEX	Yes	Yes	No	No
RECOVER ERROR RANGE	No	No	Yes	Yes
RECOVER INDEX	Yes	Yes	No	No
# RECOVER INDEX TOCOPY or # TOLOGPOINT	No	No	No	No
RECOVER TABLESPACE (no options)	No	No	Yes	Yes
RECOVER TABLESPACE TOCOPY or TORBA	No	No	No	No
REORG INDEX	Yes	Yes	No	No
REORG TABLESPACE UNLOAD CONTINUE or PAUSE	No	No	No	No
REORG TABLESPACE UNLOAD ONLY or EXTERNAL	Yes	Yes	Yes	Yes
REPAIR DUMP or VERIFY	Yes	Yes	Yes	Yes
REPAIR LOCATE INDEX PAGE REPLACE	Yes	Yes	No	No
REPAIR LOCATE KEY or RID DELETE or REPLACE	No	No	No	Yes
REPAIR LOCATE TABLESPACE PAGE REPLACE	No	No	Yes	Yes
REPORT	Yes	Yes	Yes	Yes
RUNSTATS	Yes	Yes	Yes	Yes
STOSPACE	Yes	Yes	Yes	Yes
UNLOAD	Yes	Yes	Yes	Yes

Reviewing RUNSTATS output

RUNSTATS alters tables and columns in the catalog tables. When you specify REPORT YES, RUNSTATS also generates a report of statistics that are gathered during processing.

RUNSTATS sets the following columns to -1 for table spaces that are defined as LARGE:
- CARD in SYSTABLES
- CARD in SYSINDEXPART
- FAROFFPOS in SYSINDEXPART
- NEAROFFPOS in SYSINDEXPART
- FIRSTKEYCARD in SYSINDEXES
- FULLKEYCARD in SYSINDEXES

Index statistics and table space statistics: Table 101 shows the catalog tables that RUNSTATS updates depending on the value of the UPDATE option, the value of the HISTORY option, and the source of the statistics (table space, partition, index or LOB table space).

Table 101. Catalog tables that RUNSTATS updates

Keyword	UPDATE option	HISTORY option	Catalog table that RUNSTATS updates
TABLESPACE	UPDATE ALL	HISTORY ALL [4]	SYSTABLESPACE SYSTABLEPART[1] SYSTABLES[1] SYSTABSTATS[1,2] SYSLOBSTATS[3]
TABLESPACE	UPDATE ALL	HISTORY ACCESSPATH	SYSTABLESPACE SYSTABLES[1] SYSTABSTATS[1,2]
TABLESPACE	UPDATE ALL	HISTORY SPACE	SYSTABLEPART[1] SYSLOBSTATS[3]
TABLESPACE	UPDATE ACCESSPATH[2]	HISTORY ALL [4]	SYSTABLESPACE SYSTABLES SYSTABSTATS[2]
TABLESPACE	UPDATE ACCESSPATH[2]	HISTORY ACCESSPATH	SYSTABLESPACE SYSTABLES SYSTABSTATS[2]
TABLESPACE	UPDATE ACCESSPATH[2]	HISTORY SPACE	none
TABLESPACE	UPDATE SPACE[2]	HISTORY ALL [4]	SYSTABLEPART SYSLOBSTATS SYSTABLES
TABLESPACE	UPDATE SPACE[2]	HISTORY ACCESSPATH	none
TABLESPACE	UPDATE SPACE[2]	HISTORY SPACE	SYSTABLEPART SYSLOBSTATS SYSTABLES
TABLE	UPDATE ALL	HISTORY ALL [4]	SYSCOLUMNS SYSCOLSTATS[2]
TABLE	UPDATE ALL	HISTORY ACCESSPATH	SYSCOLUMNS SYSCOLSTATS[2]
TABLE	UPDATE ALL	HISTORY SPACE	none
TABLE	UPDATE ACCESSPATH	HISTORY ALL [4]	SYSCOLUMNS SYSCOLSTATS[2]
TABLE	UPDATE ACCESSPATH	HISTORY ACCESSPATH	SYSCOLUMNS SYSCOLSTATS[2]
TABLE	UPDATE ACCESSPATH	HISTORY SPACE	none

Table 101. Catalog tables that RUNSTATS updates (continued)

Keyword	UPDATE option	HISTORY option	Catalog table that RUNSTATS updates
INDEX	UPDATE ALL	HISTORY ALL [4]	SYSCOLUMNS SYSCOLDIST SYSCOLDISTSTATS[2] SYSCOLSTATS[2] SYSINDEXES SYSINDEXPART SYSINDEXSTATS[2]
INDEX	UPDATE ALL	HISTORY ACCESSPATH	SYSCOLUMNS SYSCOLDIST SYSCOLDISTSTATS[2] SYSCOLSTATS[2] SYSINDEXPART SYSINDEXSTATS[2]
INDEX	UPDATE ALL	HISTORY SPACE	SYSINDEXES
INDEX	UPDATE ACCESSPATH	HISTORY ALL [4]	SYSCOLUMNS SYSCOLDIST SYSCOLDISTSTATS[2] SYSCOLSTATS SYSINDEXES SYSINDEXSTATS[2]
INDEX	UPDATE ACCESSPATH	HISTORY ACCESSPATH	SYSCOLUMNS SYSCOLDIST SYSCOLDISTSTATS[2] SYSCOLSTATS SYSINDEXES SYSINDEXSTATS[2]
INDEX	UPDATE ACCESSPATH	HISTORY SPACE	SYSINDEXES
INDEX	UPDATE SPACE	HISTORY ALL [4]	SYSINDEXPART SYSINDEXES[5]
INDEX	UPDATE SPACE	HISTORY ACCESSPATH	none
INDEX	UPDATE SPACE	HISTORY SPACE	SYSINDEXPART SYSINDEXES[5]

Notes:

1. Not applicable if the specified table space is a LOB table space.
2. Only updated for partitioned objects. When you run RUNSTATS against single partitions of an object, RUNSTATS uses the partition-level statistics to update the aggregate statistics for the entire object. These partition-level statistics are contained in the following catalog tables:
   - SYSCOLSTATS
   - SYSCOLDISTSTATS
   - SYSTABSTATS
   - SYSINDEXSTATS
3. Applicable only when the specified table space is a LOB table space.
4. When HISTORY NONE is specified, none of the catalog history tables are updated.
5. Only the SPACEF and STATSTIME columns are updated.

Access path statistics

The catalog table columns that are listed in Table 102, Table 103, Table 104, Table 105, Table 106, and Table 107 are used by DB2 to select access paths to data during the bind process. These columns are updated by RUNSTATS with the UPDATE ACCESSPATH or UPDATE ALL options. Refer to Part 5 (Volume 2) of *DB2 Administration Guide* for more information about these columns.

These tables do not describe information about LOB columns because DB2 does not use those statistics for access path selection. For information about what values in these columns indicate for LOBs, see Appendix F of *DB2 SQL Reference*.

A value in the "Use" column indicates whether information about the DB2 catalog column is General-use Programming Interface and Associated Guidance Information (G) or Product-sensitive Programming Interface and Associated Guidance Information (S), as defined in "Programming interface information" on page 906.

Table 102 lists the columns in SYSTABLES that DB2 uses to select access paths. These columns are updated by RUNSTATS with the UPDATE ACCESSPATH or UPDATE ALL options unless the statistics in the SYSTABSTATS table have been manually updated to -1. In this case, the columns in SYSTABLES are not updated after RUNSTATS PART UPDATE ALL is run.

Table 102. SYSTABLES catalog columns that DB2 uses to select access paths

SYSTABLES Column name	Column description	Use
CARDF	Total number of rows in the table.	S
NPAGES	Total number of pages on which rows of this table are included.	S
NPAGESF	Total number of pages that are used by the table.	S
PCTROWCOMP	Percentage of rows compressed within the total number of active rows in the table.	S

Table 103 lists the columns in SYSTABSTATS that DB2 uses to select access paths. These columns are updated by RUNSTATS with the UPDATE ACCESSPATH or UPDATE ALL options.

Table 103. SYSTABSTATS catalog columns that DB2 uses to select access paths

SYSTABSTATS Column name	Column description	Use
CARDF	Total number of rows in the partition.	S
NPAGES	Total number of pages on which rows of this partition are included.	S

Table 104 lists the columns in SYSCOLUMNS that DB2 uses to select access paths. These columns are updated by RUNSTATS with the UPDATE ACCESSPATH or UPDATE ALL options.

Table 104. SYSCOLUMNS catalog columns that DB2 uses to select access paths

SYSCOLUMNS Column name	Column description	Use
COLCARDF	Estimated number of distinct values for the column. For an indicator column, this value is the number of LOBs that are not null and whose lengths are greater than zero. The value is -1 if statistics have not been gathered. The value is -2 for columns of an auxiliary table.	S

Table 104. SYSCOLUMNS catalog columns that DB2 uses to select access paths (continued)

SYSCOLUMNS Column name	Column description	Use
HIGH2KEY	Second highest value of the column. Blank if statistics have not been gathered or if the column is an indicator column or a column of an auxiliary table. If the column has a non-character data type, the data might not be printable. This column can be updated.	S
LOW2KEY	Second lowest value of the column. Blank if statistics have not been gathered or if the column is an indicator column or a column of an auxiliary table. If the column has a non-character data type, the data might not be printable. This column can be updated.	S

Table 105 lists the columns in SYSCOLDIST that DB2 uses to select access paths. These columns are updated by RUNSTATS with the UPDATE ACCESSPATH or UPDATE ALL options.

Table 105. SYSTCOLDIST catalog columns that DB2 uses to select access paths

SYSCOLDIST Column name	Column description	Use
CARDF	The number of distinct values for the column group. This number is valid only for cardinality key column statistics. (A C in the TYPE column indicates that cardinality statistics were gathered.)	S
COLGROUPCOLNO	Identifies the set of columns that are associated with the key column statistics.	S
COLVALUE	Actual index column value that is being counted for distribution index statistics.	S
FREQUENCYF	Percentage of rows, multiplied by 100, that contain the values that are specified in COLVALUE.	S
NUMCOLUMNS	The number of columns that are associated with the key column statistics.	G

Table 106 lists the columns in SYSTABLESPACE that DB2 uses to select access paths. These columns are updated by RUNSTATS with the UPDATE ACCESSPATH or UPDATE ALL options.

Table 106. SYSTABLESPACE catalog columns that DB2 uses to select access paths

SYSTABLESPACE Column name	Column description	Use
NACTIVE or NACTIVEF	Number of active pages in the table space; shows the number of pages that are accessed if a record cursor is used to scan the entire file. The value is -1 if statistics have not been gathered.	S

Table 107 on page 576 lists the columns in SYSINDEXES that DB2 uses to select access paths. These columns are updated by RUNSTATS with the UPDATE ACCESSPATH or UPDATE ALL options.

Table 107. SYSINDEXES catalog columns that DB2 uses to select access paths

SYSINDEXES Column name	Column description	Use
CLUSTERRATIOF	A number between 0 and 1 that, when multiplied by 100, gives the percentage of rows that are in clustering order. For example, a value of 1 indicates that all rows are in clustering order. A value of .87825 indicates that 87.825% of the rows are in clustering order.	S
CLUSTERING	Indication of whether CLUSTER was specified when the index was created.	G
FIRSTKEYCARDF	Number of distinct values of the first key column.	S
FULLKEYCARDF	Number of distinct values of the full key.	S
NLEAF	Number of leaf pages in the index.	S
NLEVELS	Number of levels in the index tree.	S

Space statistics (columns for tuning information)

The catalog table columns that are listed in Table 108, Table 109, Table 110 on page 577, Table 111 on page 577, Table 112 on page 579, Table 113 on page 579, Table 114 on page 579 Table 115 on page 579, Table 116 on page 582, and Table 117 on page 582 are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options. The information in these columns helps database administrators assess the status of a particular table space or index.

A value in the ″Use″ column indicates whether information about the DB2 catalog column is General-use Programming Interface and Associated Guidance Information (G) or Product-sensitive Programming Interface and Associated Guidance Information (S), as defined in “Programming interface information” on page 906.

Table 108 lists the columns in SYSTABLESPACE that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options

Table 108. SYSTABLESPACE catalog columns that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options.

SYSTABLESPACE Column name	Column description	Use
AVGROWLEN	Average length of rows for the tables in the table space.	G

Table 109 lists the columns in SYSTABLES that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options.

Table 109. SYSTABLES catalog columns that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options

SYSTABLES Column name	Column description	Use
AVGROWLEN	Average length of rows for the tables in the table space.	G

Table 110 on page 577 lists the columns in SYSTABLES_HIST that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options.

Table 110. SYSTABLES_HIST catalog columns that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options

SYSTABLES_HIST Column name	Column description	Use
AVGROWLEN	Average length of rows for the tables in the table space.	G

Table 111 lists the columns in SYSTABLEPART that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options.

Table 111. SYSTABLEPART catalog columns that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options

SYSTABLEPART column name	Column description	Use
AVGROWLEN	Average length of rows for the tables in the table space.	G
CARDF	Total number of rows in the table space or partition, or number of LOBs in the table space if the table space is a LOB table space. The value is -1 if statistics have not been gathered.	G
DSNUM	Number of data sets.	G
EXTENTS	Number of data set extents.	G
NEARINDREF	Number of rows that are relocated near their original page. For more information about NEARINDREF, see the description of FARINDREF.	S
FARINDREF	Number of rows that are relocated far from their original page. If an update operation increases the length of a record by more than the amount of available space in the page in which it is stored, the record is moved to another page. Until the table space is reorganized, the record requires an additional page reference when it is accessed. The sum of NEARINDREF and FARINDREF is the total number of such records. For nonsegmented table spaces, a page is considered "near" the present page if the two page numbers differ by 16 or fewer; otherwise, it is "far from" the present page. For segmented table spaces, a page is considered "near" the present page if the two page numbers differ by (SEGSIZE * 2) or less. Otherwise, it is "far from" its original page. A record that is relocated near its original page tends to be accessed more quickly than one that is relocated far from its original page.	S

Table 111. SYSTABLEPART catalog columns that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options (continued)

SYSTABLEPART column name	Column description	Use
PAGESAVE	Percentage of pages that are saved in the table space or partition as a result of using data compression. For example, a value of 25 indicates a savings of 25%, so that the required pages are only 75% of what would be required without data compression. The value is 0 if no savings from using data compression are likely, or if statistics have not been gathered. The value can be negative if using data compression causes an increase in the number of pages in the data set. This calculation includes the overhead bytes for each row, the required bytes for the dictionary, and the required bytes for the current FREEPAGE and PCTFREE specification for the table space and partition. This calculation is based on an average row length, and the result varies depending on the actual lengths of the rows.	S
PERCACTIVE	Percentage of space that is occupied by rows of data from active tables. The value is -1 if statistics have not been gathered. The value is -2 if the table space is a LOB table space. This value is influenced by the PCTFREE and the FREEPAGE parameters on the CREATE TABLESPACE statement and by unused segments of segmented table spaces.	S
PERCDROP	For nonsegmented table spaces, the percentage of space that is occupied by rows of data from dropped tables. For segmented table spaces, this value is zero. After reorganization, this value is always zero. Space that is occupied by dropped tables is reclaimed by reorganization. Hence, this figure is one indicator of when a table space should be reorganized.	S
SPACE	The number of kilobytes of space that is currently allocated for all extents. A value of -1 indicates that the data set is defined with the DEFINE NO attribute, and the first insert operation has not occurred.	G
SPACEF	The number of kilobytes of disk storage.	G
PQTY (user-managed)	The primary space allocation in 4-KB blocks for the data set.	G
SQTY (user-managed)	The secondary space allocation in 4-KB blocks for the data set, in small integer format.	G
SECQTYI (user-managed)	The secondary space allocation in 4-KB blocks for the data set, in integer format.	G

Table 112 on page 579 lists the columns in SYSTABLEPART_HIST that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options.

Table 112. SYSTABLEPART_HIST that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options

SYSTABLEPART_HIST Column name	Column description	Use
AVGROWLEN	Average length of rows for the tables in the table space.	G

Table 113 lists the columns in SYSINDEXES that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options.

Table 113. SYSINDEXES catalog columns that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options

SYSINDEXES column name	Column description	Use
AVGKEYLEN	Average length of keys within the index. The value is –1 if statistics have not been gathered.	G

Table 114 lists the columns in SYSINDEXES_HIST that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options.

Table 114. SYSINDEXES_HIST catalog columns that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options

SYSINDEXES_HIST column name	Column description	Use
AVGKEYLEN	Average length of keys within the index. The value is –1 if statistics have not been gathered.	G

Table 115 lists the columns in SYSINDEXPART that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options.

Table 115. SYSINDEXPART catalog columns that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options

SYSINDEXPART column name	Column description	Use
AVGKEYLEN	Average length of keys within the index. The value is –1 if statistics have not been gathered.	G
CARDF	Number of rows that the index or partition refers to.	S
DSNUM	Number of data sets.	G
EXTENTS	Number of data set extents.	G

Table 115. SYSINDEXPART catalog columns that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options (continued)

SYSINDEXPART column name	Column description	Use
FAROFFPOSF	Number of times that accessing a different, "far-off" page would be necessary when accessing all the data records in index order. Each time that DB2 accesses a far-off page, accessing the "next" record in index order would probably require I/O activity. For nonsegmented table spaces, a page is considered far-off from the present page if the two page numbers differ by 16 or more. For segmented table spaces, a page is considered far-off from the present page if the two page numbers differ by SEGSIZE * 2 or more. Together, NEAROFFPOS and FAROFFPOS indicate how well the index follows the cluster pattern of the table space. For a clustering index, NEAROFFPOS and FAROFFPOS approach a value of 0 as clustering improves. A reorganization should bring them nearer their optimal values; however, if a nonzero FREEPAGE value is specified on the CREATE TABLESPACE statement, the NEAROFFPOS after reorganization reflects the table on which the index is defined. Do not expect optimal values for nonclustering indexes. FAROFFPOS is not applicable for the index on an auxiliary table (-1).	S
LEAFNEAR	Number of leaf pages that are located physically near previous leaf pages for successive active leaf pages.	S
LEAFFAR	Number of leaf pages that are located physically far away from previous leaf pages for successive active leaf pages that are accessed in an index scan.	S

Table 115. SYSINDEXPART catalog columns that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options (continued)

SYSINDEXPART column name	Column description	Use
NEAROFFPOSF	Number of times that accessing a different, "near-off" page would be necessary when accessing all the data records in index order. Each time that DB2 accesses a near-off page, accessing the "next" record in index order would probably require I/O activity. For more information about NEAROFFPOS, see the description of FAROFFPOS. NEAROFFPOS is incremented if the current indexed row is not on the same or next data page of the previous indexed row, and if the distance between the two data pages does not qualify for FAROFFPOS. For nonsegmented table spaces, a page is considered near-off from the present page if the difference between the two page numbers is greater than or equal to 2, and less than 16. For segmented table spaces, a page is considered near-off from the present page if the difference between the two page numbers is greater than or equal to 2, and less than SEGSIZE * 2. A nonzero value in the NEAROFFPOS field after a REORG might be attributed to the number of space map pages that are contained in the segmented table space. NEAROFFPOS is not applicable for the index on an auxiliary table (-1).	S
LEAFDIST	100 times the average distance in page IDs between successive leaf pages during a sequential access of the index. This value indicates how well an index is organized. The value is at its lowest immediately after the index has been reorganized. Changes increase the value; and you can reduce it again by reorganizing the index, either explicitly or as part of a general table space reorganization.	S
PSEUDO_DEL_ ENTRIES	Number of pseudo-deleted keys. Pseudo-deleted keys are keys that are marked for deletion, but the data still resides in the index space, and the space can be reused. The space can be reclaimed by running REORG INDEX.	S
SPACE	The number of kilobytes of space that is currently allocated for all extents. A value of -1 indicates that the data set is defined with the DEFINE NO attribute, and the first insert operation has not occurred.	G
SPACEF	The number of kilobytes of disk storage.	G
PQTY (user-managed)	The primary space allocation in 4-KB blocks for the data set.	G
SQTY (user-managed)	The secondary space allocation in 4-KB blocks for the data set, in small integer format.	G

Table 115. SYSINDEXPART catalog columns that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options (continued)

SYSINDEXPART column name	Column description	Use
SECQTYI (user-managed)	The secondary space allocation in 4-KB blocks for the data set, in integer format.	G

Table 116 lists the columns in SYSINDEXPART_HIST that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options.

Table 116. SYSINDEXPART_HIST catalog columns that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options

SYSINDEXPART_HIST column name	Column description	Use
AVGKEYLEN	Average length of keys within the index. The value is –1 if statistics have not been gathered.	G

Table 117 lists the columns in SYSLOBSTATS that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options.

Table 117. SYSLOBSTATS catalog columns that are updated by RUNSTATS with the UPDATE SPACE or UPDATE ALL options

SYSLOBSTATS column name	Column description	Use
AVGSIZE	The average size of a LOB in the LOB table space.	G
FREESPACE	The number of kilobytes of available space in the LOB table space, up to the highest used RBA.	S
ORGRATIO	The percentage of organization in the LOB table space. A value of 100 indicates perfect organization of the LOB table space. A value of 1 indicates that the LOB table space is disorganized. A value of 0.00 indicates that the LOB table space is totally disorganized. An empty LOB table space has an ORGRATIO value of 100.	S

After running RUNSTATS

After running RUNSTATS with the UPDATE ACCESSPATH option, the UPDATE SPACE option, or the UPDATE ALL option, rebind any application plans that use the tables or indexes so that they use the new statistics.

Sample RUNSTATS control statements

Example 1: Updating catalog statistics for a table space while allowing changes. The following control statement specifies that the RUNSTATS utility is to update the catalog with statistics for table space DSN8D81A.DSN8S81E and all of its associated tables and indexes. When updating the table statistics, RUNSTATS is to sample 25% of the rows. The SHRLEVEL change option indicates that DB2 is to permit other processes to make changes while this utility is executing.

```
//STEP1    EXEC DSNUPROC,UID='IUJQU225.RUNSTA',TIME=1440,
//         UTPROC='',
//         SYSTEM='DSN',DB2LEV=DB2A
//UTPRINT  DD  SYSOUT=*
//SYSIN    DD *
RUNSTATS TABLESPACE DSN8D81A.DSN8S81E
   TABLE(ALL) SAMPLE 25
    INDEX(ALL)
   SHRLEVEL CHANGE
```

Example 2: Updating index statistics. The following control statement specifies that RUNSTATS is to update the catalog statistics for index DSN8810.XEMPL1.

```
RUNSTATS INDEX (DSN8810.XEMPL1)
```

Example 3: Updating index statistics while prohibiting updates. The following control statement specifies that RUNSTATS is to update the catalog statistics for indexes XEMPL1 and XEMPL2. DB2 does not permit other processes to change the table space that is associated with XEMPL1 and XEMPL2 (table space DSN8S81E) while this utility is executing. This restricted access is the default behavior.

```
RUNSTATS INDEX (DSN8810.XEMPL1,DSN8810.XEMPL2)
```

Example 4: Updating statistics for columns in several tables. The following control statement specifies that RUNSTATS is to update the catalog statistics for the following columns in table space DSN8D81P.DSN8S81C:

- All columns in the TCONA and TOPTVAL tables
- The LINENO and DSPLINE columns in the TDSPTXT table

```
RUNSTATS TABLESPACE(DSN8D81P.DSN8S81C)
         TABLE (TCONA)
         TABLE (TOPTVAL) COLUMN(ALL)
         TABLE (TDSPTXT) COLUMN(LINENO,DSPLINE)
```

Example 5: Updating all statistics for a table space. The following control statement specifies that RUNSTATS is to update all catalog statistics (table space, tables, columns, and indexes) for table space DSN8D81P.DSN8S81C.

```
RUNSTATS TABLESPACE(DSN8D81P.DSN8S81C) TABLE INDEX
```

Example 6: Updating statistics that are used for access path selection and generating a report. The following control statement specifies that RUNSTATS is to update the catalog with **only** the statistics that are collected for access path selection. The utility is to report **all** statistics for the table space and route the report to SYSPRINT.

```
RUNSTATS TABLESPACE DSN8D81A.DSN8S81E
  REPORT YES
  UPDATE ACCESSPATH
```

Example 7: Updating all statistics and generating a report. The following control statement specifies that RUNSTATS is to update the catalog with **all** statistics (access path and space) for table space DSN8D81A.DSN8S81E. The utility is also to report the collected statistics and route the report to SYSPRINT.

```
RUNSTATS TABLESPACE DSN8D81A.DSN8S81E
  REPORT YES
  UPDATE ALL
```

Example 8: Reporting statistics without updating the catalog. The following control statement specifies that RUNSTATS is to collect statistics for table space DSN8D81A.DSN8S81E and route the report to SYSPRINT. The utility is not to update the catalog with the collected statistics.

```
RUNSTATS TABLESPACE DSN8D81A.DSN8S81E
  REPORT YES
  UPDATE NONE
```

Example 9: Updating statistics for a partition. The following control statement specifies that RUNSTATS is to update the statistics for the first partition of table space DSN8D81A.DSN8S81E and the first partition of the DSN8810.XEMP1 index.

```
RUNSTATS TABLESPACE DSN8D81A.DSN8S81E PART 1 INDEX(DSN8810.XEMP1 PART 1)
```

Example 10: Updating catalog and history tables and reporting all statistics. The following control statement specifies that RUNSTATS is to update the catalog tables and history catalog tables with all statistics for table space DB0E0101.TL0E0101 (including related indexes and columns). The utility is to report the collected statistics and route the statistics to SYSPRINT.

```
RUNSTATS TABLESPACE DBOE0101.TLOE0101
   INDEX
   TABLE
   REPORT YES
   UPDATE ALL
   HISTORY ALL
```

Example 11: Updating statistics on frequently occurring values. Assume that the SYSADM.IXNP1 index is defined on four columns: NP1, NP2, NP3, and NP4. The following control statement specifies that RUNSTATS is to update the statistics for index SYSADM.IXNPI.

The KEYCARD option indicates that the utility is to collect cardinality statistics for column NP1, column set NP1 and NP2, and column set NP1, NP2, and NP3, and column set NP1, NP2, NP3, and NP4. The FREQVAL option and its associated parameters indicate that RUNSTATS is also to collect the 5 most frequently occurring values on column NP1 (the first key column of the index), and the 10 most frequently occurring values on the column set NP1 and NP2 (the first two key columns of the index). The utility is to report the collected statistics and route the statistics to SYSPRINT.

```
RUNSTATS INDEX (SYSADM.IXNPI)
           KEYCARD
           FREQVAL NUMCOLS 1 COUNT 5
           FREQVAL NUMCOLS 2 COUNT 10
           REPORT YES
```

Example 12: Updating distribution statistics for a group of specified columns in a table. The following control statement specifies that RUNSTATS is to update statistics for the columns EMPLEVEL, EMPGRADE, and EMPSALARY in table DSN8810.DEPT (in table space DSN8D81A.DSN8S81E). The statement uses the COLGROUP keyword to group these columns. RUNSTATS is to collect the cardinality of this column group and store the cardinality in the SYSCOLDIST catalog table.

```
RUNSTATS TABLESPACE DSN8D81A.DSN8S81E
TABLE(DSN8810.DEPT)
COLGROUP (EMPLEVEL,EMPGRADE,EMPSALARY)
```

Example 13: Updating distribution statistics for specific columns and retrieving the most frequently occurring values. The following control statement specifies that RUNSTATS is to update statistics for the columns EMPLEVEL, EMPGRADE, and EMPSALARY in table DSN8810.DEPT. The FREQVAL and COUNT options indicate that RUNSTATS is to collect the 10 most frequently occurring values for each column. The values are to be stored in the SYSCOLDIST and SYSCOLDISTSTATS catalog tables.

```
RUNSTATS TABLESPACE DSN8D81A.DSN8S81E
TABLE(DSN8810.DEPT)
COLGROUP(EMPLEVEL,EMPGRADE,EMPSALARY) FREQVAL COUNT 10
```

Example 14: Updating distribution statistics for specific columns in a table and retrieving the least frequently occurring values. The following control statement specifies that RUNSTATS is to update statistics for the columns EMPLEVEL, EMPGRADE, and EMPSALARY in table DSN8810.DEPT. The FREQVAL and COUNT options indicate that RUNSTATS is to collect the 15 least frequently occurring values for each column. The values are to be stored in the SYSCOLDIST and SYSCOLDISTSTATS catalog tables.

```
RUNSTATS TABLESPACE DSN8D81A.DSN8S81E
TABLE(DSN8810.DEPT)
COLGROUP(EMPLEVEL,EMPGRADE,EMPSALARY) FREQVAL COUNT 15 LEAST
```

Example 15: Updating distribution statistics for specific columns in a table space and retrieving the most and least frequently occurring values. The following control statement specifies that RUNSTATS is to update statistics for the columns EMPLEVEL, EMPGRADE, and EMPSALARY in table DSN8810.DEPT. The FREQVAL and COUNT options indicate that RUNSTATS is to collect the 10 most frequently occurring values for each column and the 10 least frequently occurring values for each column. The values are to be stored in the SYSCOLDIST and SYSCOLDISTSTATS catalog tables.

```
RUNSTATS TABLESPACE DSN8D81A.DSN8S81E
TABLE(DSN8810.DEPT)
COLGROUP(EMPLEVEL,EMPGRADE,EMPSALARY) FREQVAL COUNT 10 BOTH
```

Example 16: Updating statistics for an index and retrieving the most and least frequently occurring values. The following control statement specifies that RUNSTATS is to collect the 10 most frequently occurring values and the 10 least frequently occurring values for the first key column of index ADMF001.IXMA0101. The KEYCARD option indicates that the utility is also to collect all the distinct values in all the key column combinations. A set of messages is sent to SYSPRINT and all collected statistics are updated in the catalog.

```
RUNSTATS INDEX(ADMF001.IXMA0101)
          KEYCARD
FREQVAL NUMCOLS 1 COUNT 10 BOTH
REPORT YES UPDATE ALL
```

Example 17: Invalidating statements in the dynamic statement cache for a table space without generating report statistics. The following control statement specifies that RUNSTATS is to invalidate statements in the dynamic statement cache for table space DSN8D81A.DSN8S81E. However, RUNSTATS is not to collect or report statistics or update the catalog.

```
RUNSTATS TABLESPACE DSN8D81A.DSN8S81E
REPORT NO
UPDATE NONE
```

Chapter 30. STOSPACE

The STOSPACE utility updates DB2 catalog columns that indicate how much space is allocated for storage groups and related table spaces and indexes.

For a diagram of STOSPACE syntax and a description of available options, see "Syntax and options of the STOSPACE control statement." For detailed guidance on running this utility, see "Instructions for running STOSPACE" on page 588.

Output: The output from STOSPACE consists of new values in a number of catalog tables. See "Reviewing STOSPACE output" on page 591 for a list of columns and tables that STOSPACE updates.

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:
- STOSPACE privilege
- SYSCTRL or SYSADM authority

Execution phases of STOSPACE: The STOSPACE utility operates in these phases:

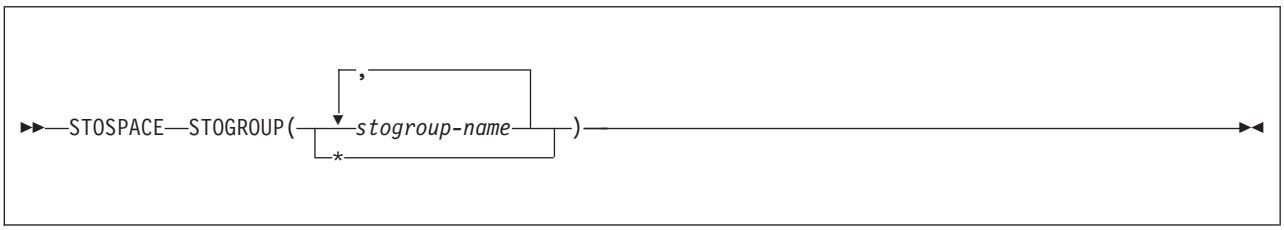
Phase	Description
UTILINIT	Performs initialization
STOSPACE	Gathers space information and updates catalog
UTILTERM	Performs cleanup

The following topics provide additional information:
- "Syntax and options of the STOSPACE control statement"
- "Instructions for running STOSPACE" on page 588
- "Concurrency and compatibility for STOSPACE" on page 591
- "Reviewing STOSPACE output" on page 591
- "Sample STOSPACE control statement" on page 591

Syntax and options of the STOSPACE control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram



Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

STOGROUP
Identifies the storage groups that are to be processed.

(*stogroup-name***, ...)**
Specifies the name of a storage group. You can use a list of from one to 255 storage group names. Separate items in the list by commas, and enclose them in parentheses.

*****
Indicates that all storage groups are to be processed.

Instructions for running STOSPACE

To run STOSPACE, you must:

1. Prepare the necessary data sets, as described in "Data sets that STOSPACE uses."
2. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for STOSPACE, see "Sample STOSPACE control statement" on page 591.)
3. Prepare a utility control statement that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks" on page 589.
4. Check the compatibility rules in "Concurrency and compatibility for STOSPACE" on page 591 if you want to run other jobs concurrently on the same target objects.
5. Plan for restart if the STOSPACE job doesn't complete, as described in "Terminating or restarting STOSPACE" on page 591.
6. Run STOSPACE by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Data sets that STOSPACE uses

Table 118 lists the data sets that STOSPACE uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 118. Data sets that STOSPACE uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement.	Yes

Table 118. Data sets that STOSPACE uses (continued)

Data set	Description	Required?
SYSPRINT	Output data set for messages.	Yes

The following object is named in the utility control statement and does not require a DD statement in the JCL:

Storage group
Object that is to be reported.

Creating the control statement

Create the utility control statement for the STOSPACE job. See "Syntax and options of the STOSPACE control statement" on page 587 for STOSPACE syntax and option descriptions. See "Sample STOSPACE control statement" on page 591 for examples of STOSPACE usage.

Instructions for specific tasks

To perform the following tasks, specify the options and values for those tasks in your utility control statement:

Ensuring availability of objects that STOSPACE requires

For each specified storage group, STOSPACE looks at the SYSIBM.SYSTABLESPACE and SYSIBM.SYSINDEXES catalog tables to determine which objects belong to that storage group. For each object, the amount of allocated space is determined from an appropriate VSAM catalog. Hence the table spaces and indexes do not need to be available to DB2 when STOSPACE is running; only the DB2 catalog and appropriate VSAM catalogs are required. However, to gain access to the VSAM catalog, the utility must have available to it the database definition (DBD) for the objects that are involved. This access requires that the appropriate database, table spaces, and index spaces not be in the stopped state.

Obtaining statistical information with STOSPACE

Table 119 lists statistical information that the STOSPACE utility records and that is useful for making space allocation decisions.

Table 119. DB2 catalog data that STOSPACE collects

Catalog table	Column name	Column description
SYSTABLESPACE	SPACEF	Number of kilobytes of storage that are allocated to the table space
SYSTABLEPART	SPACEF	Number of kilobytes of storage that are allocated to the table space partition
SYSINDEXES	SPACEF	Number of kilobytes of storage that are allocated to the index
SYSINDEXPART	SPACEF	Number of kilobytes of storage that are allocated to the index partition
SYSSTOGROUP	SPACEF	Number of kilobytes of storage that are allocated to the storage group

Table 119. DB2 catalog data that STOSPACE collects (continued)

Catalog table	Column name	Column description
SYSSTOGROUP	STATSTIME	Time when STOSPACE was last run on a particular storage group

When DB2 storage groups are used in the creation of table spaces and indexes, DB2 defines the data sets for them. The STOSPACE utility permits a site to monitor the disk space that is allocated for the storage group.

STOSPACE does not accumulate information for more than one storage group. If a partitioned table space or index space has partitions in more than one storage group, the information in the catalog about that space comes from only the group for which STOSPACE was run.

When you run the STOSPACE utility, the SPACEF column of the catalog represents the high-allocated RBA of the VSAM linear data set. Use the value in the SPACEF column to project space requirements for table spaces, table space partitions, index spaces, and index space partitions over time. Use the output from the Access Method Services LISTCAT command to determine which table spaces and index spaces have allocated secondary extents. When you find these, increase the primary quantity value for the data set, and run the REORG utility.

For information about space utilization in the DSN8S81E table space in the DSN8D81A database, first run the STOSPACE utility, and then execute the following SQL statement:

General-use Programming Interface

```
EXEC SQL
  SELECT SPACE
  FROM SYSIBM.SYSTABLESPACE
  WHERE NAME = 'DSN8S81E
  AND DBNAME = 'DSN8D81A'
ENDEXEC
```

End of General-use Programming Interface

Alternatively, you can use TSO to look at data set and pack descriptions.

To update SYSIBM.SYSSTOGROUP for storage group DSN8G810, as well as SYSIBM.SYSTABLESPACE and SYSIBM.SYSINDEXES, for every table space and index that belongs to DSN8G810, use the following utility control statement:

```
STOSPACE STOGROUP DSN8G810
```

Analyzing the values in a SPACE or SPACEF column

The value in a SPACE or SPACEF column represents total allocated space, not only the space that is allocated on the current list of volumes in the storage groups. (If the value is too large to fit in the SPACE column, the SPACEF column is used.) You can delete volumes from a storage group even though space on those volumes is still allocated to DB2 table spaces or indexes. Deletion of a volume from a storage group prevents future allocations; it does not withdraw a current allocation.

Running STOSPACE on user-defined spaces

For user-defined spaces, STOSPACE does not record any statistics.

Terminating or restarting STOSPACE

You can terminate a STOSPACE utility job with the TERM UTILITY command if you have submitted the job or have SYSOPR, SYSCTRL, or SYSADM authority.

You can restart a STOSPACE utility job, but it starts from the beginning again. For guidance in restarting online utilities, see "Restarting an online utility" on page 43.

Concurrency and compatibility for STOSPACE

STOSPACE does not set a utility restrictive state on the target object.

STOSPACE can run concurrently with any utility on the same target object. However, because STOSPACE updates the catalog, concurrent STOSPACE utility jobs or other concurrent applications that update the catalog might cause timeouts and deadlocks.

You can use the STOSPACE utility on storage groups that have objects within
temporary databases.

Reviewing STOSPACE output

The output from STOSPACE consists of updated values in the columns and tables in the following list. In each case, an amount of space is given in kilobytes (KB).

- SPACE [4] in SYSIBM.SYSINDEXES shows the amount of space that is allocated to indexes. If the index is not defined using STOGROUP, or if STOSPACE has not been executed, the value is zero.
- SPACE[4] in SYSIBM.SYSTABLESPACE shows the amount of space that is allocated to table spaces. If the table space is not defined using STOGROUP, or if STOSPACE has not been executed, the value is zero.
- SPACE[4] in SYSIBM.SYSINDEXPART shows the amount of space that is allocated to index partitions. If the partition is not defined using STOGROUP, or if STOSPACE has not been executed, the value is zero.
- SPACE[4] in SYSIBM.SYSTABLEPART shows the amount of space that is allocated to table partitions. If the partition is not defined using STOGROUP, or if STOSPACE has not been executed, the value is zero.
- SPACE[4] in SYSIBM.SYSSTOGROUP shows the amount of space that is allocated to storage groups.
- STATSTIME in SYSIBM.SYSSTOGROUP shows the timestamp for the time at which STOSPACE was last executed.

Sample STOSPACE control statement

Example 1: Updating catalog SPACE columns for a particular storage group. The following control statement specifies that the STOSPACE utility is to update the
| catalog SPACE or SPACEF columns for storage group DSN8G810 and any related table spaces and indexes.

```
//STEP1    EXEC DSNUPROC,UID='FUAUU330.STOSPCE',
//         UTPROC='',
//         SYSTEM='DSN'
//SYSIN    DD *
STOSPACE STOGROUP DSN8G810
//*
```

4. If the value is too large to fit in the SPACE column, the SPACEF column is updated.

Example 2: Specifying a storage group name that contains spaces. If the name of the storage group that you want STOSPACE to process contains spaces, enclose the entire storage group name in single quotation marks. Parentheses are optional. The following statements are correct ways to specify a storage group with the name THIS IS STOGROUP.1.ON.E:

```
STOSPACE STOGROUP('THIS IS STOGROUP.1.ONE')

STOSPACE STOGROUP 'THIS IS STOGROUP.1.ONE'
```

Example 3: Updating catalog SPACE columns for all storage groups. The following control statement specifies that the STOSPACE utility is to update the catalog SPACE or SPACEF columns for all storage groups.

```
STOSPACE STOGROUP *
```

Example 4: Updating catalog SPACE columns for several storage groups. The following control statement specifies that the STOSPACE utility is to update the catalog SPACE or SPACEF columns for storage groups DSN8G810 and DSN8G81U.

```
STOSPACE STOGROUP(DSN8G810, DSN8G81U)
```

Chapter 31. TEMPLATE

The TEMPLATE utility control statement lets you allocate data sets, without using JCL DD statements, during the processing of a LISTDEF list. In its simplest form, the TEMPLATE control statement defines the data set naming convention. You can also write TEMPLATE control statements so that they contain allocation parameters that define data set size, location, and attributes.

Templates enable you to standardize data set names across the DB2 subsystem and to easily identify the data set type when you use variables in the data set name. These variables are listed in "Option descriptions" on page 596.

The TEMPLATE control statement uses the z/OS DYNALLOC macro (SVC 99) to perform data set allocation. Therefore, the facility is constrained by the limitations of this macro and by the subset of DYNALLOC that is supported by TEMPLATE. See *z/OS MVS Programming: Assembler Services Guide* for more details.

Output: The TEMPLATE control statement generates a dynamic allocation template with an assigned name for later reference.

Authorization required: No privileges are required to execute this control statement. When a TEMPLATE is referenced by a specific utility, privileges are checked at that time.

Execution phases of TEMPLATE: The TEMPLATE control statement executes entirely in the UTILINIT phase, which performs setup for the subsequent utility.

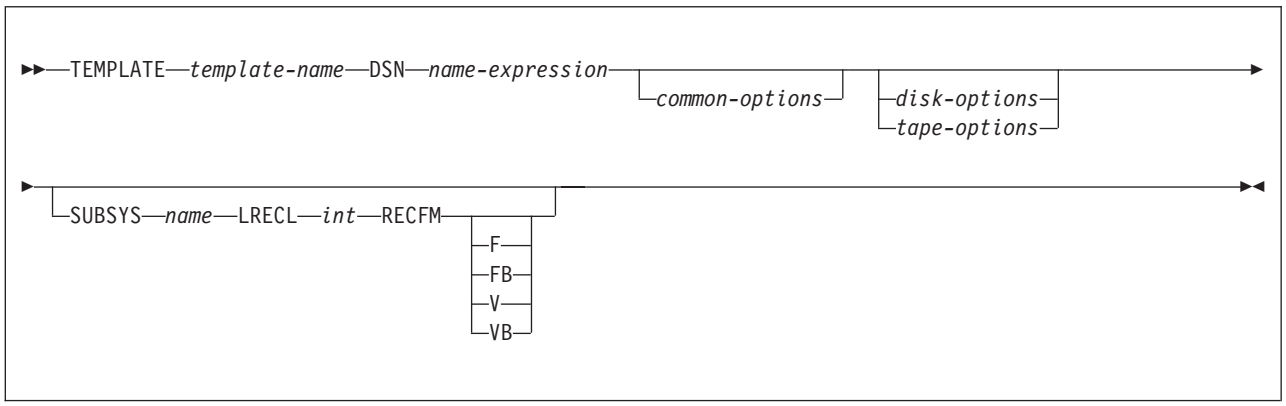
The following topics provide additional information:

- "Syntax and options of the TEMPLATE control statement "
- "Instructions for using TEMPLATE" on page 606
- "Concurrency and compatibility for TEMPLATE" on page 609
- "Sample TEMPLATE control statements" on page 609

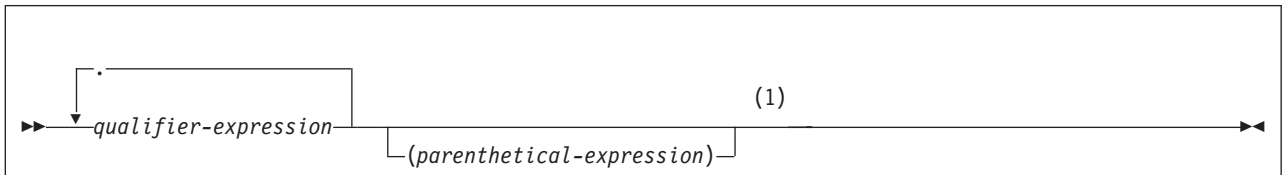
Syntax and options of the TEMPLATE control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram



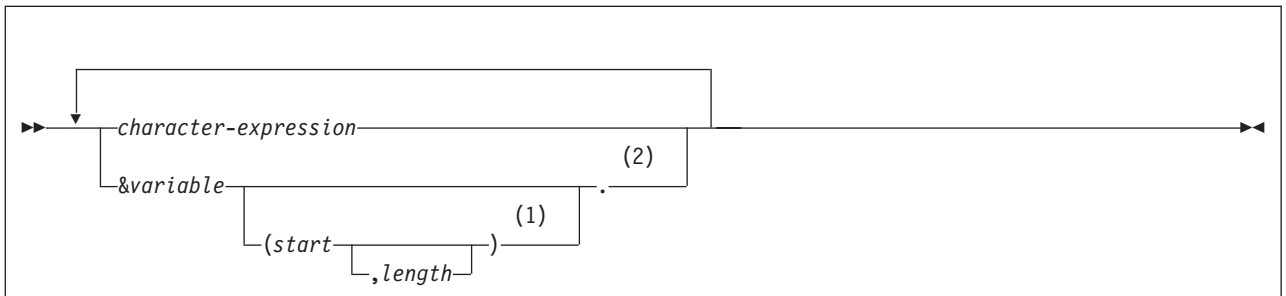
name-expression:



Notes:

1 The entire name-expression represents one character string and cannot contain any blanks.

qualifier-expression:

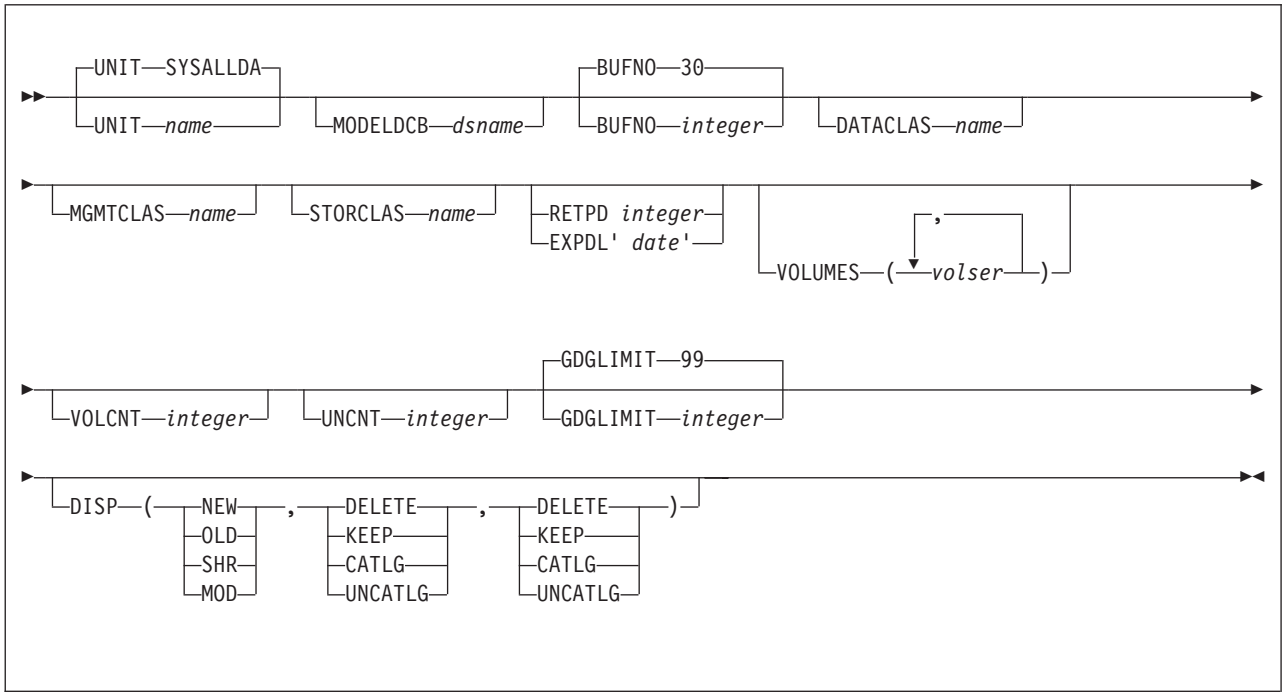


Notes:

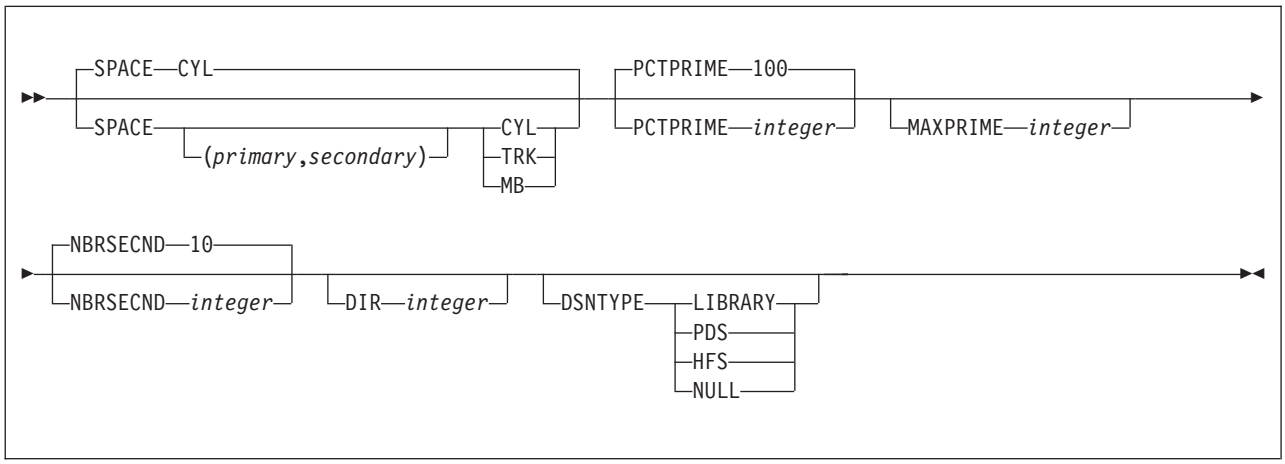
1 If you use substring notation, the entire DSN operand must be enclosed in single quotation marks. For example, the DSN operand `'P&PA(4,2).'` uses substring notation, so it is enclosed in single quotation marks.

2 The `&PA.` variable cannot be used more than once.

common-options:



disk-options:



tape-options:

Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

TEMPLATE *template-name*

Defines a data set allocation template and assigns to the template a name, *template-name*, for subsequent reference on a DB2 utility control statement. The *template-name* can have up to eight alphanumeric characters and must begin with an alphabetic character.

The *template-name* is followed by keywords that control the allocation of tape and disk data sets. A single TEMPLATE statement cannot have both disk options and tape options. The UNIT keyword specifies a generic unit name that is defined on your system. This value is used to determine if a disk or tape data set is being allocated. All other keywords specified on the TEMPLATE control statement must be consistent with the specified unit type.

DSN *name-expression*

Specifies the template for the z/OS data set name. You can specify the data set name, *name-expression*, by using symbolic variables, non-variable alphanumeric, or national characters, or any combination of these characters. The resulting name must adhere to the z/OS data set naming rules, including those rules about name length, valid characters, name structure and qualifier length.

Data set names consists of a series of qualifiers, *qualifier-expression*, that are separated by a period (.) and an optional parenthetical expression. No imbedded blanks are allowed.

If the DSN name operand contains any special characters, it must be enclosed in single quotation marks. For example, in the following TEMPLATE statement, the DSN operand contains the parentheses special character, so the entire operand is enclosed in single quotation marks:

```
TEMPLATE X DSN 'A.GDG.VERSION(+1)'
```

Parentheses around the DSN name operand are optional. They are used in the following DSN specification:

```
DSN(&DB..&TS..D&DATE.)
```

character-expression

Specifies the data set name or part of the data set name by using non-variable alphanumeric or national characters.

parenthetical-expression

Specifies part of the data set name by using non-variable alphanumeric or national characters that are enclosed in parentheses. For example, the expressions `Q1.Q2.Q3(member)` and `Q1.Q2.Q3(+1)` use valid parenthetical expressions.

&variable. Specifies the data set name or part of the data set name by using symbolic variables. See Table 120 on page 597, Table 121 on page 598, Table 122 on page 598, and Table 123 on page 599 for a list of variables that can be used.

Each symbolic variable is substituted with its related value at execution time to form a specific data set name. When used in a

DSN expression, substitution variables begin with an ampersand sign (&) and end with a period (.), as in the following example:

```
DSN &DB..&TS..D&JDATE..COPY&ICTYPE.
```

Using numeric variables alone generates an invalid data set qualifier for all numeric-type variables (all date or time-type variables, and others, such as &SEQ. or &PART.). These variables must be preceded by character constants to form valid DSN qualifiers. The following examples are valid specifications:

```
P&PART.
D&DATE.
```

Some substitution variables are invalid if you use TEMPLATE with an incompatible utility. For example, ICTYPE is not meaningful if the TEMPLATE statement is used with LOAD SYSDISC. Other variables assume default values when their values are not known. For example, &PART. becomes 00000 for non-partitioned objects.

You can also use substring notation for data set name variables. This notation can help you keep the data set name from exceeding the 44 character maximum. If you use substring notation, the entire DSN operand must be enclosed in single quotation marks. To specify a substring, use the form &variable(start). or &variable(start,length).

start
Specifies the substring's starting byte location within the current variable base value at the time of execution. *start* must be an integer from 1 to 128.

length
Specifies the length of the substring. If you specify *start* but do not specify *length*, *length*, by default, is the number of characters from the *start* character to the last character of the variable value at the time of execution. For example, given a five-digit base value, `&PART(4).` specifies the fourth and fifth digits of the value. *length* must be an integer that does not cause the substring to extend beyond the end of the base value. For more examples of variable substring notation, see "Sample TEMPLATE control statements" on page 609.

Table 120 contains a list of JOB variables and their descriptions.

Table 120. JOB variables

Variable	Description
`&JOBNAME.` or `&JO.`	The z/OS job name.
`&STEPNAME.` or `&ST.`	The z/OS step name. This variable might be needed if data set names from two different job steps conflict.
`&USERID.` or `&US.`	The user ID of the person that is running the utility. The value is 1 to 8 characters long.
`&UTILID.` or `&UT.`	The utility ID truncated to eight characters and checked for invalid DSN characters.
`&SSID.` or `&SS.`	Subsystem ID (non-data sharing) or group attach name (data sharing).

Table 121 on page 598 contains a list of UTILITY variables and

their descriptions.

Table 121. UTILITY variables

Variable	Description
&ICTYPE. or &IC.	Single-character image copy type. This variable is valid only for image copy templates. The substitution is governed by whether a full image copy (F), an incremental image copy (I), or a CHANGELIMIT image copy (C) is specified by the user.
&UTILNAME. or &UN.	Special values are assigned to some utilities: CHECKD for CHECK DATA, CHECKI for CHECK INDEX, CHECKL for CHECK LOB, REORGI for REORG INDEX, and REORGT for REORG TABLESPACE. Utility names that are longer than eight characters are truncated to eight characters.
&SEQ. or &SQ.	Sequence number of the list item in the list.
&LOCREM. or &LR.	Indicator of whether *ddname* is for the local site (COPYDDN) or the recovery site (RECOVERYDDN). Single character L is used when the utility defines a COPYDDN *ddname*. The single character R is used when the utility defines a RECOVERYDDN *ddname*. You can replicate the SYSCOPY ICBACKUP column information by using both the &LOCREM. and &PRIBAC. variables. This variable is valid only for image copy templates.
&PRIBAC. or &PB.	Indicator of whether *ddname* is for the primary (*ddname1*) or backup (*ddname2*) copy data set. Single character P is used when the utility defines a *ddname1*. The single character B is used when the utility defines a *ddname2*. You can replicate the SYSCOPY ICBACKUP column information by using both the &LOCREM. and &PRIBAC. variables. This variable is valid only for image copy templates.

Table 122 contains a list of OBJECT variables and their descriptions.

Table 122. OBJECT variables

Variable	Description
&LIST. or &LI.	The name of the list that is defined by using the LISTDEF control statement and that is referenced on the same control statement as this TEMPLATE. This variable is used with COPY FILTERDDN templates. All objects in the list are copied to one data set, which makes &TS. and &IS. meaningless.
&DB.	Database name.
&TS.[1]	Table space name.
&IS.[1]	Index space name.
&SN.[1]	Space name (table space or index space).
&PART. or &PA.[2]	Five-digit partition number, padded with leading zeros.

Table 122. OBJECT variables (continued)

Variable	Description

Notes:

1. When you specify the &TS., &IS., or &SN. variables in a template that is used by an UNLOAD statement with BLOBF, CLOBF, or DBCLOBF, DB2 substitutes the name of the table space that stores the LOB column value, not the base table space name. This substitution enables DB2 to generate unique data set names for each LOB column with partitioned table spaces.
2. Use the &PA. variable when processing LISTDEF lists with the PARTLEVEL keyword or data-partitioned secondary indexes. Otherwise, DB2 could generate duplicate data set names.

Table 123 contains a list of DATE and TIME variables. and their descriptions.

Table 123. DATE and TIME variables

Variable	Description
`&DATE.` or `&DT.`	YYYYDDD
`&TIME.` or `&TI.`	HHMMSS
`&JDATE.` or `&JU.`	YYYYDDD
`&YEAR.` or `&YE.`	YYYY portion of `&DATE.`
`&MONTH.` or `&MO.`	MM
`&DAY.` or `&DA.`	DD
`&JDAY.` or `&JD.`	DDD portion of `&DATE.`
`&HOUR.` or `&HO.`	HH portion of `&TIME.`
`&MINUTE.` or `&MI.`	MM portion of `&TIME.`
`&SECOND.` or `&SC.`	SS portion of `&TIME.`
`&UNIQ.` or `&UN.`	Unique eight characters that DB2 derives from the system clock at the time of allocation. This set of characters begins with an alphabetical character and is followed by seven alphabetical or numeric characters.

Note: All date and time values are set by using the STCK instruction, and they reflect the date and time value in Greenwich Mean Time (GMT). With the exception of `&UNIQ.` and `&UN.` DATE and TIME values are captured in the UTILINIT phase of each utility and remain constant until the utility terminates.`&UNIQ.` and `&UN.` are assigned a unique value for each allocation.

SUBSYSname Specifies the MVS BATCHPIPES SUBSYSTEM name. The SUBSYS operand must be a valid BATCHPIPES SUBSYSTEM name and must not exceed eight characters in length. When SUBSYS is specified, LRECL and RECFM are required. When SUBSYS is specified, TEMPLATE keywords that are not compatible with SUBSYS (such as UNIT) are ignored.

Restriction:: When using BATCHPIPES, TEMPLATE with the SUBSYS keyword, the utility cannot be restarted and the LOAD DISCARDDN keyword is not supported.

LRECLint Specifies the record length of the MVS BATCHPIPES SUBSYSTEM file. There is no default value and this option is required when SUBSYS is specified.

RECFM Specifies the record format of the MVS BATCHPIPES SUBSYSTEM file. The valid values are F, FB, V, or VB. There is no default value and this option is required when SUBSYS is specified.

common-options (apply to both disk and tape)

UNIT Specifies the device-number, device-type (generic), or group-name for the data set. All other TEMPLATE keywords are validated based on the specified type of unit (disk or tape). The **default** is **SYSALLDA**.

MODELDCB *dsname*
Specifies the name of the data set on which the template is based. DCB information is read from this model data set.

If this is not coded and a GDG Base does not already exist, the TEMPLATE module will pass below parameter as the default for creating the GDG data set: DEFINE GDG(NAME(data set name)LIM(99)SCR).

BUFNO *integer*
Specifies the number of BSAM buffers. The specified value must be in the range from 0 to 99. The **default** is **30**.

DATACLAS *name*
Specifies the SMS data class. The *name* value must be a valid SMS data class and must not exceed eight characters in length.

The data set is cataloged if DATACLAS is specified. If this option is omitted, no DATACLAS is specified to SMS.

MGMTCLAS *name*
Specifies the SMS management class. The *name* value must be a valid SMS management class and must not exceed eight characters in length.

The data set is cataloged if MGMTCLAS is specified. If this option is omitted, no MGMTCLAS is specified to SMS.

STORCLAS *name*
Specifies the SMS storage class. The *name* value must be a valid SMS storage class and must not exceed eight characters in length.

The data set is cataloged if STORCLAS is specified. If this option is omitted, no STORCLAS is specified to SMS.

RETPD *integer* Specifies the retention period in days for the data set. The integer value must be in the range from 0 to 9999.

If DATACLAS, MGMTCLAS, or STORCLAS is specified, the class definition might control the retention. RETPD cannot be specified with EXPDL..

EXPDL *'date'* Specifies the expiration date for the data set, in the form YYYYDDD, where YYYY is the four-digit year, and DDD is the three-digit Julian day. The 'date' value must be enclosed by single quotation marks.

If DATACLAS, MGMTCLAS, or STORCLAS is specified, the class definition might control the retention. EXPDL cannot be specified with RETPD.

VOLUMES *(vol1,vol2,...)*
Specifies a list of volume serial numbers for this allocation. If the data set is not cataloged the list is truncated, if necessary, when it is stored in SYSIBM.SYSCOPY. The specified number of volumes cannot exceed the specified or default value of VOLCNT.

The first volume must contain enough space for the primary space allocation.

If an individual volume serial-number contains leading zeros, it must be enclosed in single quotation marks.

VOLCNT *(integer)*
Specifies the maximum number of volumes that an output data set might require. The specified value must be between 0 and 255. The **default** for tape templates is **95**. For disk templates, the utility does not set a default value. Operating system defaults apply.

UNCNT *integer*
Specifies the number of devices that are to be allocated. The specified value must in the range from 0 to 59.

If UNIT specifies a specific device number, the value of UNCNT must either be 1 or be omitted.

GDGLIMIT *(integer)*
Specifies the number of entries that are to be created in a GDG base if a GDG DSN is specified and the base does not already exist. If a GDG base does not already exist and you do not want to define one, specify a GDGLIMIT of zero (0).

| The **default** value is **99**. The integer value must be in the range
| from 0 to 255.

DISP *(value1, value2, value3)*
Specifies the data set disposition by using three positional parameters: status, normal-termination, and abnormal-termination. All three parameters must be specified.

status
Standard z/OS values are allowed: NEW, OLD, SHR, MOD.

normal-termination
Standard z/OS values are allowed: DELETE, KEEP, CATLG, UNCATLG.

abnormal-termination
Standard z/OS values are allowed: DELETE, KEEP, CATLG, UNCATLG.

Default values for DISP vary, depending on the utility and the data set that is being allocated. Defaults for restarted utilities also differ from default values for new utility executions. Default values are shown in Table 124 on page 602 and Table 125 on page 602.

Table 124 on page 602 shows the data dispositions for dynamically allocated data sets for new utility executions.

Table 124. Data dispositions for dynamically allocated data sets for new utility executions

ddname	CHECK DATA	CHECK INDEX or CHECK LOB	COPY	COPY-TOCOPY	LOAD	MERGE-COPY	REBUILD INDEX	REORG INDEX	REORG TABLE-SPACE	UNLOAD
SYSREC	Ignored	Ignored	Ignored	Ignored	OLD KEEP KEEP	Ignored	Ignored	Ignored	NEW CATLG CATLG	NEW CATLG CATLG
SYSDISC	Ignored	Ignored	Ignored	Ignored	NEW CATLG CATLG	Ignored	Ignored	Ignored	NEW CATLG CATLG	Ignored
SYSPUNCH	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	NEW CATLG CATLG	NEW CATLG CATLG
SYSCOPY	Ignored	Ignored	NEW CATLG CATLG	Ignored	NEW CATLG CATLG	NEW CATLG CATLG	Ignored	Ignored	NEW CATLG CATLG	Ignored
SYSCOPY2	Ignored	Ignored	NEW CATLG CATLG	Ignored	NEW CATLG CATLG	NEW CATLG CATLG	Ignored	Ignored	NEW CATLG CATLG	Ignored
SYSRCPY1	Ignored	Ignored	NEW CATLG CATLG	Ignored	NEW CATLG CATLG	NEW CATLG CATLG	Ignored	Ignored	NEW CATLG CATLG	Ignored
SYSRCPY2	Ignored	Ignored	NEW CATLG CATLG	Ignored	NEW CATLG CATLG	NEW CATLG CATLG	Ignored	Ignored	NEW CATLG CATLG	Ignored
SYSUT1	NEW DELETE CATLG	NEW DELETE CATLG	Ignored	Ignored	NEW DELETE CATLG	Ignored	NEW DELETE CATLG	NEW CATLG CATLG	NEW DELETE CATLG	Ignored
SORTOUT	NEW DELETE CATLG	Ignored	Ignored	Ignored	NEW DELETE CATLG	Ignored	Ignored	NEW DELETE CATLG	NEW DELETE CATLG	Ignored
SYSMAP	Ignored	Ignored	Ignored	Ignored	NEW CATLG CATLG	Ignored	Ignored	Ignored	Ignored	Ignored
SYSERR	NEW CATLG CATLG	Ignored	Ignored	Ignored	NEW CATLG CATLG	Ignored	Ignored	Ignored	Ignored	Ignored
FILTERDDS	Ignored	Ignored	NEW DELETE DELETE	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored

Table 125 shows data dispositions for dynamically allocated data sets on RESTART.

Table 125. Data dispositions for dynamically allocated data sets on RESTART

ddname	CHECK DATA	CHECK INDEX or CHECK LOB	COPY	COPY-TOCOPY	LOAD	MERGE-COPY	REBUILD INDEX	REORG INDEX	REORG TABLE-SPACE	UNLOAD
SYSREC	Ignored	Ignored	Ignored	Ignored	OLD KEEP KEEP	Ignored	Ignored	Ignored	MOD CATLG CATLG	MOD CATLG CATLG
SYSDISC	Ignored	Ignored	Ignored	Ignored	MOD CATLG CATLG	Ignored	Ignored	Ignored	MOD CATLG CATLG	Ignored
SYSPUNCH	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	MOD CATLG CATLG	MOD CATLG CATLG
SYSCOPY	Ignored	Ignored	MOD CATLG CATLG	Ignored	MOD CATLG CATLG	MOD CATLG CATLG	Ignored	Ignored	MOD CATLG CATLG	Ignored
SYSCOPY2	Ignored	Ignored	MOD CATLG CATLG	Ignored	MOD CATLG CATLG	MOD CATLG CATLG	Ignored	Ignored	MOD CATLG CATLG	Ignored
SYSRCPY1	Ignored	Ignored	MOD CATLG CATLG	Ignored	MOD CATLG CATLG	MOD CATLG CATLG	Ignored	Ignored	MOD CATLG CATLG	Ignored

Table 125. Data dispositions for dynamically allocated data sets on RESTART (continued)

ddname	CHECK DATA	CHECK INDEX or CHECK LOB	COPY	COPY-TOCOPY	LOAD	MERGE-COPY	REBUILD INDEX	REORG INDEX	REORG TABLE-SPACE	UNLOAD
SYSRCPY2	Ignored	Ignored	MOD CATLG CATLG	Ignored	MOD CATLG CATLG	MOD CATLG CATLG	Ignored	Ignored	MOD CATLG CATLG	Ignored
SYSUT1	MOD DELETE CATLG	MOD DELETE CATLG	Ignored	Ignored	MOD DELETE CATLG	Ignored	MOD DELETE CATLG	MOD CATLG CATLG	MOD DELETE CATLG	Ignored
SORTOUT	MOD DELETE CATLG	Ignored	Ignored	Ignored	MOD DELETE CATLG	Ignored	Ignored	MOD DELETE CATLG	MOD DELETE CATLG	Ignored
SYSMAP	Ignored	Ignored	Ignored	Ignored	MOD CATLG CATLG	Ignored	Ignored	Ignored	Ignored	Ignored
SYSERR	MOD CATLG CATLG	Ignored	Ignored	Ignored	MOD CATLG CATLG	Ignored	Ignored	Ignored	Ignored	Ignored
FILTERDDS	Ignored	Ignored	NEW DELETE DELETE	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored

End of common-options (apply to both disk and tape)

disk-options

SPACE *(primary,secondary)*

Specifies the z/OS disk space allocation parameters in the range from 1 to 1677215. If you specify *(primary,secondary)* value, these values are used instead of the DB2-calculated values. When specifying primary and secondary quantities, you must either specify both values or omit both values.

Use the MAXPRIME option to set an upper limit on the *primary* quantity.

CYL

Specifies that allocation quantities, if present, are to be expressed in cylinders and that allocation is to occur in cylinders. If SPACE CYL is specified, without *(primary,secondary)*, the DB2-calculated quantities are allocated in cylinders by using 3390 quantities for byte conversion.

The **default** is **CYL**.

TRK

Specifies that, in the absence of values for *(primary,secondary)*, the DB2-calculated quantities are to be allocated in tracks by using 3390 disk drive quantities for byte conversion. If the amount calculated is greater than one cylinder, the TRK keyword is ignored and the data set is allocated in cylinders (CYL).

MB

Specifies that allocation quantities, if present, are to be expressed in megabytes, and that allocation is to occur in records. One megabyte is 1 048 576 bytes. If SPACE MB is specified, without *(primary,secondary)*, the DB2-calculated quantities are allocated in records by using the average record length for the data set.

See "Default space calculations" on page 608 for default space calculations for each utility data set.

PCTPRIME *integer*
Specifies the percentage of the estimated required space that is to be obtained as the primary quantity. The **default** is **100**.

Use the MAXPRIME option to set the upper limit of this value for large objects.

MAXPRIME *integer*
Specifies the maximum allowable primary space allocation, expressed in cylinders (CYL). This value constrains the *primary* space value and the PCTPRIME calculation, as well as the size of each secondary allocation.

NBRSECND *integer*
Specifies the division of secondary space allocations. After the primary space is allocated, an amount of space equal to the estimated required space is divided into the specified number of secondary allocations. The integer value must be in the range from 1 to 10. The **default** value is **10**.

DIR *integer* Specifies the number of 256-byte records that are to be allocated for the directory of a new partitioned data set. You must specify this operand if you are allocating a new partitioned data set.

If the template is being used in a UNLOAD statement with BLOBF, CLOBF, or DBCLOBF and you specify a DSNTYPE of LIBRARY or PDS, but do not specify DIR, DB2 calculates the number of 256-byte records to allocate by dividing the estimated number of records by 20.

DSNTYPE Specifies the type of data set to be allocated.

LIBRARY
Specifies that a partitioned data set extended (PDSE) is to be allocated.

PDS
Specifies that a partitioned data set (PDS) is to be allocated.

HFS
Specifies that a hierarchical file system (HFS) file is to be allocated.

NULL
Specifies a null file. Use this value for a template with UNLOAD CLOBF, BLOBF, or DBCLOBF to unload a null LOB value. In this case, the unload data set contains a null file name.

If you omit DSNTYPE, the type of data set is determined by other data set attributes, the data class for the data set, or an installation default.

End of disk-options

tape-options

STACK Specifies whether output data sets are to be stacked contiguously on the same tape volumes.

NO
Specifies that output data sets are not to be stacked contiguously on tape. The **default** is **NO**.

YES
Specifies that similar output data sets are to be stacked as successive files on one logical tape volume, where a logical tape volume can consist of a multi-volume aggregate. Within one utility execution, output data sets are stacked on a logical tape volume of the same usage type. For example, local primary image copies are stacked separately from local backup image copies.

Restriction: Do not use the STACK YES option for concurrent copies (copies that are made by the COPY utility with the CONCURRENT option).

To preserve parallel processing, parallel tasks are written to different tape volumes. The specific volume to which the data set is written can vary, depending on the number of output data sets that are being produced, the number of parallel processes that are requested, and the number of tape units that are available to the job step.

The data sets and utilities for which the STACK YES option are supported are listed in Table 126. ″Yes″ indicates that the specified utility supports tape stacking for the specified data set. ″No″ indicates that the specified utility does not support tape stacking for the specified data set. ″Ignored″ indicates that the specified data set does not apply to the specified utility.

Table 126. Supported data sets for tape stacking

ddname	CHECK DATA	CHECK INDEX or CHECK LOB	COPY	COPY-TOCOPY	LOAD	MERGE-COPY	REBUILD INDEX	REORG INDEX	REORG TABLE-SPACE	UNLOAD
SYSREC	Ignored	Ignored	Ignored	Ignored	No	Ignored	Ignored	Ignored	Yes	Yes
SYSDISC	Ignored	Ignored	Ignored	Ignored	No	Ignored	Ignored	Ignored	Yes	Ignored
SYSPUNCH	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	Yes	Yes
SYSCOPY	Ignored	Ignored	Yes	Yes	No	Yes	Ignored	Ignored	Yes	Ignored
SYSCOPY2	Ignored	Ignored	Yes	Yes	No	Yes	Ignored	Ignored	Yes	Ignored
SYSRCPY1	Ignored	Ignored	Yes	Yes	No	Yes	Ignored	Ignored	Yes	Ignored
SYSRCPY2	Ignored	Ignored	Yes	Yes	No	Yes	Ignored	Ignored	Yes	Ignored
SYSUT1	No	No	Ignored	Ignored	No	Ignored	No	No	No	Ignored
SORTOUT	No	Ignored	Ignored	Ignored	No	Ignored	Ignored	No	No	Ignored
SYSMAP	Ignored	Ignored	Ignored	Ignored	No	Ignored	Ignored	Ignored	Ignored	Ignored
SYSERR	No	Ignored	Ignored	Ignored	No	Ignored	Ignored	Ignored	Ignored	Ignored
FILTERDDS	Ignored	Ignored	No	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored

TRTCH Specifies the track recording technique for magnetic tape drives that have improved data recording capability.

NONE
Specifies that the TRTCH specification is to be eliminated from dynamic allocation. The **default** is **NONE**.

COMP
Specifies that data is to be written in compacted format.

NOCOMP
Specifies that data is to be written in standard format.

End of tape-options

Instructions for using TEMPLATE

Some DB2 utilities produce data sets during execution. These data sets are referenced in utility control statements by a set of DD name keywords and are specified in detail in the corresponding JCL.

As an alternative to using JCL to specify the data sets, you can use the TEMPLATE utility control statement to dynamically allocate utility data sets. Options of the TEMPLATE utility allow you to specify the following information:
- The data set naming convention
- DFSMS parameters
- Disk or tape allocation parameters

You can specify a template in the SYSIN data set, immediately preceding the utility control statement that references it, or in one or more TEMPLATE libraries.

A TEMPLATE library is a data set that contains only TEMPLATE utility control statements. You can specify a TEMPLATE data set DD name by using the TEMPLATEDD option of the OPTIONS utility control statement. This specification applies to all subsequent utility control statements until the end of input or until DB2 encounters a new OPTIONS TEMPLATEDD(*ddname*) specification.

Any template that is defined within SYSIN overrides another template definition of the same name in a TEMPLATE data set.

TEMPLATE utility control statements enable you to standardize data set allocation and the utility control statements that reference those data sets, which reduces the need to customize and alter utility job streams.

Key TEMPLATE operations

Like both LISTDEF and OPTIONS utility control statements, a TEMPLATE control statement performs a setup operation in preparation for use by another utility. When the control statement is processed, the information is saved under the template name for the duration of the job step. You can reference it as though it were an output data set DD name by substituting the template name for the DD name on most utility control statements. If a DD name and a TEMPLATE name conflict, the DD statement is used for allocation, and the TEMPLATE is ignored. Minimally, a TEMPLATE statement consists of a name (similar to a DD name) and a data set naming convention. If nothing else is specified, DB2 calculates the required data set size and uses default data set attributes that are appropriate to the data set that is being created. DB2 then allocates a disk data set with these defaults.

The required TEMPLATE statement might look something like the following TEMPLATE statement:

```
TEMPLATE tmp1  DSN(DB2.&TS..D&JDATE..COPY&ICTYPE.&LOCREM.&PRIBAC.)
    VOLUMES(vol1,vol2,vol3)
LISTDEF payroll INCLUDE TABLESPACE PAYROLL.*
                INCLUDE INDEXSPACE PAYROLL.*IX
                EXCLUDE TABLESPACE PAYROLL.TEMP*
                EXCLUDE INDEXSPACE PAYROLL.TMPIX*
COPY LIST payroll  COPYDDN(tmp1,tmp1) RECOVERYDDN(tmp1,tmp1)
```

See “Syntax and options of the TEMPLATE control statement ” on page 593 for details.

Creating data set names

The data set naming convention that is specified on the DSN option of each TEMPLATE statement must be appropriate for the data set that is being created and coordinated with the other templates and DD statements in the same job step. The data set name must be both unique and meaningful. DB2 does not check that the data set names are unique until the execution of the utility that references the template. Ensure that the data set names are unique when you define the data set naming convention on the TEMPLATE control statement. Follow these guidelines when developing template names:

- Use a combination of static characters, national characters, and the provided variable names to form valid z/OS data set qualifiers. Normal z/OS rules apply. Variables that produce numeric values must be preceded by either a static character or a character variable. All qualifiers must start with an alphabetic character. The qualifiers must consist of a maximum of eight characters and a maximum of 44 characters for the entire data set name. To help comply with this 44 character limit, you can use variable substring notation. For more information about variable substring notation, see the description of &variable in “Option descriptions” on page 596.
- Use the two-character form of the DSN variables to save space.
- Use two consecutive periods following all variables that precede the last qualifier (one to terminate the variable, followed by a second static period to separate the qualifiers), as in the following example:

  ```
  &DB..&TS.
  ```
- Use &DB. and &TS. to relate the data set to a database object.
- Use &PART. when executing PARTLEVEL lists. Preceed the variable with a static character or a character variable to form a valid qualifier.
- Use &JO. and &ST. to eliminate conflicts with other jobs or job steps.
- Use &SS., &US., &UT., and &UN. if you have a need to know the subsystem, member, user, utility ID, or name of the utility that produced the data set.
- Use &DATE. and &TIME. or the shorter substring variations to guarantee uniqueness. Preceed the variable with a static character or a character variable to form a valid qualifier.
- Use &IC., &LR., and &PB. to identify image copy data sets. For example, the following template name would make a meaningful seven-character data set qualifier:

  ```
  COPY&IC.&LR.&PB.
  ```

Controlling data set size

You can also use the TEMPLATE syntax to specify disk space parameters. If you do not specify the SPACE keyword, DB2 estimates the size of the data set based on formulas that vary according to the utility and the data set. See “Default space calculations” on page 608 for details.

DB2 usually estimates the size of a data set based on the size of other existing data sets; however, if any of the required data sets are on tape, DB2 is unable to estimate the size. When DB2 is able to calculate size, it calculates the maximum size. This action can result in overly large data sets. DB2 always allocates data set size with the RLSE (release) option so that unused space is released on deallocation. However in some cases, the calculated size of required data sets is too large for the DYNALLOC interface to handle. In this case, DB2 issues error message DSNU1034I, you must allocate the data set by a DD statement. If the object is part of a LISTDEF list, you might need to remove it from the list and process it individually.

Database administrators can check utility control statements without executing them by using the PREVIEW function. In PREVIEW mode, DB2 expands all TEMPLATE data set names in the SYSIN DD, in addition to any data set name from the TEMPLATE DD that are referenced on a utility control statement. DB2 then prints the information to the SYSPRINT data set and halts execution. You can specify PREVIEW in one of two ways, either as a JCL PARM or on the OPTIONS PREVIEW utility control statement.

Default space calculations

Three keywords are provided to let you manage how DB2 allocates the required space for the data set. You can use these three keywords, in combination, to constrain and quantify the allocation extents.

PCTPRIME
100% of the required space estimated by DB2 is allocated as a PRIMARY quantity. If this amount of space is typically not available on a single volume, decrease PCTPRIME.

MAXPRIME
If you want an upper limit based on size, not on percentage, use MAXPRIME.

NBRSECND
After the restrictions on the PRIMARY quantity have been applied, a SECONDARY quantity equal to the estimated required space is divided into the specified number of secondary extents.

If you omit the SPACE option quantities, current data set space estimation formulas that are shown in the "Data sets that *utility* uses" sections for each online utility are implemented as default values for disk data sets.

Recommendation: Run the RUNSTATS utility with the UPDATE SPACE or UPDATE ALL option before you run any of the following utilities to improve the accuracy of the default space estimation:

- CHECK DATA
- CHECK INDEX
- CHECK LOB
- REBUILD INDEX
- REORG INDEX
- REORG TABLESPACE

Working with TAPE

The STACK keyword supports tape processing in two forms. The first form, STACK NO, supports traditional, single-file processing. The data set is written, and

the tape is rewound and repositioned or even remounted. The second form, STACK YES, lets successive files be written on a single logical tape without repositioning or remounting. Important considerations for STACK YES processing include:

- You can stack only like files on the same tape. For example, one tape might contain local primary image copies whereas another tape might contain remote primary image copies. The file types cannot be mixed.
- DB2 stacks files only within a single utility invocation. When that utility ends, the stack is terminated (meaning that the tape is rewound and unloaded). To allow stacking, use a LISTDEF list to force multiple objects to be processed under a single utility invocation.
- Parallel processing can complicate stacking. To prevent conflicts between parallel processes, use a single process to write a file to a given stack.

Working with GDGs

TEMPLATE DSN operands support both GDG absolute version references and relative references. DB2 detects the absence of a GDG base and creates it, with a limit of 99 entries, by default. Use the keyword GDGLIMIT to alter this value or prohibit this action. If you use the PREVIEW function on the OPTIONS utility control statement, DB2 displays the GDG relative version references. GDG names are restricted to 35 characters.

A model data set, as defined in the MODELDCB option, might be required to allocate GDG data sets in your environment.

Terminating or restarting TEMPLATE

You can terminate a TEMPLATE utility job by using the TERM UTILITY command if you submitted the job or have SYSOPR, SYSCTRL, or SYSADM authority.

You can restart a TEMPLATE utility job, but it starts from the beginning again. If you are restarting this utility as part of a larger job in which TEMPLATE completed successfully, but a later utility failed, do not change the TEMPLATE utility control statement, if possible. If you must change the TEMPLATE utility control statement, use caution; any changes can cause the restart processing to fail. For example, if you change the template name of a temporary work data set that was opened in an earlier phase and closed but is to be used later, the job fails. For guidance in restarting online utilities, see “Restarting an online utility” on page 43.

Concurrency and compatibility for TEMPLATE

TEMPLATE is a control statement that is used to set up an environment for another utility to follow. The template is stored until it is referenced by a specific utility. The list is expanded when it is referenced by another utility. At that time, the concurrency and compatibility restrictions of that utility apply, and the catalog tables that are necessary to expand the list must be available for read-only access.

Sample TEMPLATE control statements

Example 1: Specifying a basic template for an image copy on disk. The following TEMPLATE utility control statement defines a basic template that can be used to allocate an image copy data set. The name of the template is COPYDS. Any subsequent COPY jobs that specify this template for dynamically allocated data sets use the data set naming convention that is defined by the DSN option.

```
TEMPLATE COPYDS DSN &DB..&TS..COPY&IC.&LR.&PB..D&DATE..T&TIME.
```

Example 2: Using variable substring notation to specify data set names. The following control statement defines template CP2. Variable substring notation is used in the DSN option to define the data set naming convention.

Assume that in the year 2003 you make a full image copy of partition 00004 of table space DSN8S81D. Assume that you specify the template CP2 for the data set for the local primary copy. DB2 gives the following name to the image copy data set: DH173001.DSN8S81D.Y03.COPYLP.P004

Notice that every variable in the DSN option begins with an ampersand (&) and ends with a period (.). These ampersands and periods are not included in the data set name. Only periods that do not signal the end of a variable are included in the data set name.

```
TEMPLATE CP2 DSN 'DH173001.&SN..Y&YEAR(3)..COPY&LR.&PB..P&PART(3,3).'
    UNIT(SYSDA)
```

Example 3: Using COPY with TEMPLATE with variable substring notation. The following TEMPLATE utility control statement defines template SYSCOPY. Variable substring notation is used in the DSN option to define the data set naming convention. The subsequent COPY utility control statement specifies that DB2 is to make a local primary copy of the first partition of table space DSN8D81A.DSN8S81E. COPY is to write this image copy to a data set that is dynamically allocated according to the SYSCOPY template. In this case, the resulting data set name is DSN8D81A.DSN8S81E.P001

```
TEMPLATE SYSCOPY DSN '&DB..&TS..P&PA(3).'

COPY TABLESPACE DSN8D81A.DSN8S81E DSNUM 1 COPYDDN(SYSCOPY)
```

Notice that you can change the part variable in the DSN operand from P&PA(3). to P&PA(3,3). The resulting data set name is the same, because the length value of 3 is implied in the first specification.

Example 4: Specifying a template for tape data sets with an expiration date. The following control statement defines the TAPEDS template. Any data sets that are defined with this template are to be allocated on device number 3590-1, as indicated by the UNIT option, and are to expire on 1 January 2100, as indicated by the EXPDL option. The DSN option indicates that these data set names are to have the following three parts: database name, table space name, and date.

```
TEMPLATE TAPEDS DSN(&DB..&TS..D&DATE.)
    UNIT 3590-1  EXPDL '2100001'
```

Example 5: Specifying a disk template that gives space allocation parameters. The following control statement defines the DISK template. Any data sets that are defined with this template are to have 100 cylinders of primary disk space and 10 cylinders of secondary disk space, as indicated by the SPACE and CYL options. The DSN option indicates that the data set names are to have the following three parts: database name, table space name, and time.

```
TEMPLATE DISK DSN &DB..&TS..T&TIME.
    SPACE(100,10) CYL
```

Example 6: Specifying a disk template that uses a default size with constraints. The following control statement defines the DISK template. Because the SPACE option does not specify quantities for primary and secondary space allocation, DB2 calculates these values with the following constraint: the maximum allowable primary space allocation is 1000 cylinders. This constraint is indicated by the

MAXPRIME option. The DSN option indicates that the data set names are to have the following three parts: database name, table space name, and time.

```
TEMPLATE DISK DSN(&DB..&TS..T&TIME.)
    SPACE CYL  MAXPRIME 1000
```

Example 7: Using TEMPLATE with LISTDEF and COPY. In the following example, the LISTDEF utility control statement defines the CPY1 list. The TEMPLATE control statement then defines the TMP1 template. The COPY utility control statement then specifies that DB2 is to make local copies of the objects in the CPY1 list. DB2 is to write these copies to data sets that are dynamically allocated according to the characteristics that are defined in the TMP1 template.

```
LISTDEF CPY1 INCLUDE TABLESPACES TABLESPACE DBA906*.T*A906*
             INCLUDE INDEXSPACES COPY YES INDEXSPACE ADMF001.I?A906*
TEMPLATE TMP1 UNIT SYSDA
              DSN (DH109006.&STEPNAME..&SN..T&TIME.)
              DISP (MOD,CATLG,CATLG)
COPY LIST CPY1 COPYDDN (TMP1) PARALLEL (2) SHRLEVEL REFERENCE
```

Note: Parentheses for the DSN name-expression are optional.

Example 8: Use TEMPLATE to create a GDG data set. In the example in Figure 104, the TEMPLATE control statement defines the COPYTEMP template. The COPY utility control statement specifies that DB2 is to write a local image copy of the table space DBLT2501.TPLT2501 to a data set that is dynamically allocated according to the characteristics that are defined in the COPYTEMP template. According to the COPYTEMP template, this data set is to be named JULTU225.GDG(+1) (as indicated by the DSN option) and is to have six entries created in the GDG base (as indicated by the GDGLIMIT option). The control block information is to be the same as that in the JULTU225.MODEL data set, as indicated by the MODELDCB option.

```
//*************************************************************
//* COMMENT:   Define a model data set.                       *
//*************************************************************
//STEP1    EXEC   PGM=IEFBR14
//SYSCOPX  DD DSN=JULTU225.MODEL,DISP=(NEW,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20)),VOL=SER=SCR03,
//         DCB=(RECFM=FB,BLKSIZE=4000,LRECL=100)
//*************************************************************
//* COMMENT: GDGLIMIT(6)
//*************************************************************
//STEP2    EXEC DSNUPROC,UID='JULTU225.GDG',
//         UTPROC='',
//         SYSTEM='SSTR'
//SYSIN    DD *
   TEMPLATE COPYTEMP
            UNIT SYSDA
            DSN 'JULTU225.GDG(+1)'
            MODELDCB JULTU225.MODEL
            GDGLIMIT(6)
   COPY TABLESPACE DBLT2501.TPLT2501
        FULL YES
        COPYDDN (COPYTEMP)
        SHRLEVEL REFERENCE
/*
```

Figure 104. Example TEMPLATE and COPY statements for writing a local copy to a data set that is dynamically allocated according to the characteristics of the template.

Example 9: Using a template to copy a GDG data set to tape. In the example in Figure 105 on page 612, the OPTIONS utility control statement causes the

subsequent TEMPLATE statement to run in PREVIEW mode. In this mode, DB2 checks the syntax of the TEMPLATE statement. If DB2 determines that the syntax is valid, it expands the data set names. The OPTIONS OFF statement ends PREVIEW mode processing. The subsequent COPY utility control statement executes normally. The COPY statement specifies that DB2 is to write a local image copy of the table space DBLT4301.TPLT4301 to a data set that is dynamically allocated according to the characteristics that are defined in the COPYTEMP template. According to the COPYTEMP template, this data set is to be named JULTU243.GDG(+1) (as indicated by the DSN option) and is to be stacked on the tape volume 99543 (as indicated by the UNIT, STACK, and VOLUMES options). The data set dispositions are specified by the DISP option. The GDGLIMIT option specifies that 50 entries are to be created in a GDG base.

```
/*
//************************************************
//* COMMENT: COPY GDG DATA SET TO TAPE
//************************************************
//STEP1    EXEC DSNUPROC,UID='JULTU243.GDG',
//         UTPROC='',
//         SYSTEM='SSTR'
//SYSIN    DD *
   OPTIONS PREVIEW
   TEMPLATE COPYTEMP
            UNIT TAPE
            DSN 'JULTU243.GDG(+1)'
            VOLUMES (99543)
            GDGLIMIT(50)
            DISP(NEW,CATLG,CATLG)
            STACK YES
   OPTIONS OFF
   COPY TABLESPACE DBLT4301.TPLT4301
        FULL YES
        COPYDDN (COPYTEMP)
        SHRLEVEL REFERENCE
/*
```

Figure 105. Example job that uses OPTIONS, TEMPLATE, and COPY statements to copy a GDG data set to tape.

Example 10: Creating a template that can be used for unloading LOB objects The
TEMPLATE control statement in Figure 106 defines a template called LOBFRV. The
subsequent UNLOAD statement specifies that each CLOB in the RESUME column
is to be unloaded to files that are dynamically allocated according to the
characteristics defined for the LOBFRV template. In this case, those files are to be
partitioned data sets, as specified by the DSNTYPE option. Each data set is to have
the name UNLODTEST.*database-name*.*LOB-table-space-name*.RESUME, as specified by
the DSN option. The names of each CLOB PDS is written to the unload data set.
By default, the unload data set is defined by the SYSREC DD statement or
template.

```
#
TEMPLATE LOBFRV DSN 'UNLDTEST.&DB..&TS..RESUME'
                DSNTYPE(PDS) UNIT(SYSDA)
#
UNLOAD DATA
  FROM TABLE DSN8910.EMP_PHOTO_RESUME
  (EMPNO  CHAR(6),
   RESUME VARCHAR(255) CLOBF LOBFRV)
 SHRLEVEL CHANGE
#
```

Figure 106. Example job that creates a template that can be used for unloading LOB objects.

Chapter 32. UNLOAD

The online UNLOAD utility unloads data from one or more source objects to one or more BSAM sequential data sets in external formats. The source can be DB2 table spaces or DB2 image copy data sets. The source cannot be a concurrent copy. UNLOAD must be run on the system where the definitions of the table space and the table exists.

UNLOAD is an enhancement of the REORG UNLOAD EXTERNAL function. With UNLOAD, you can unload rows from an entire table space or select specific partitions or tables to unload. You can also select columns by using the field specification list. If a table space is partitioned, you can unload all of the selected partitions into a single data set, or you can unload each partition in parallel into physically distinct data sets.

The output records that the UNLOAD utility writes are compatible as input to the LOAD utility; as a result, you can reload the original table or different tables.

For a diagram of UNLOAD syntax and a description of available options, see "Syntax and options of the UNLOAD control statement" on page 614. For detailed guidance on running this utility, see "Instructions for running UNLOAD" on page 647.

Output: UNLOAD generates an unloaded table space or partition.

Authorization required: To execute this utility, you must use a privilege set that includes one of the following authorities:
- Ownership of the tables
- SELECT privilege on the tables
- DBADM authority for the database
- SYSADM authority
- SYSCTRL authority (catalog tables only)

If you use RACF access control with multilevel security and UNLOAD is to process a table space that contains a table that has multilevel security with row-level granularity, you must be identified to RACF and have an accessible valid security label. Each row is unloaded only if your security label dominates the data security label. If your security label does not dominate the data security label, the row is not unloaded, but DB2 does not issue an error message. For more information about multilevel security and security labels, see Part 3 of *DB2 Administration Guide*.

Execution phases of UNLOAD: The UNLOAD utility operates in these phases:

Phase	Description
UTILINIT	Performs initialization.
UNLOAD	Unloads records to sequential data sets. One pass through the input data set is made. If UNLOAD is processing a table space or partition, DB2 takes internal commits. These commits provide commit points at which the utility can be restarted in case operation should halt in this phase.
UTILTERM	Performs cleanup.

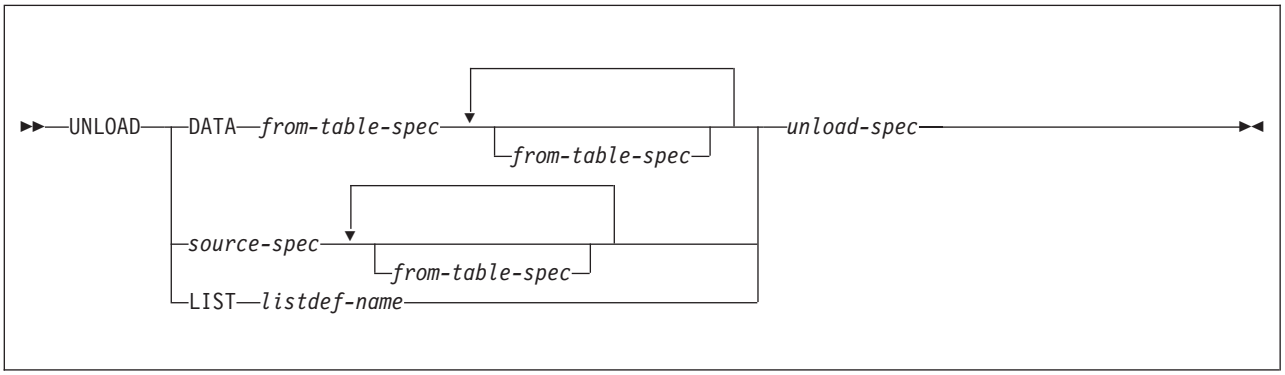
The following topics provide additional information:

- "Syntax and options of the UNLOAD control statement"
- "Instructions for running UNLOAD" on page 647
- "Concurrency and compatibility for UNLOAD" on page 660
- "Sample UNLOAD control statements" on page 662

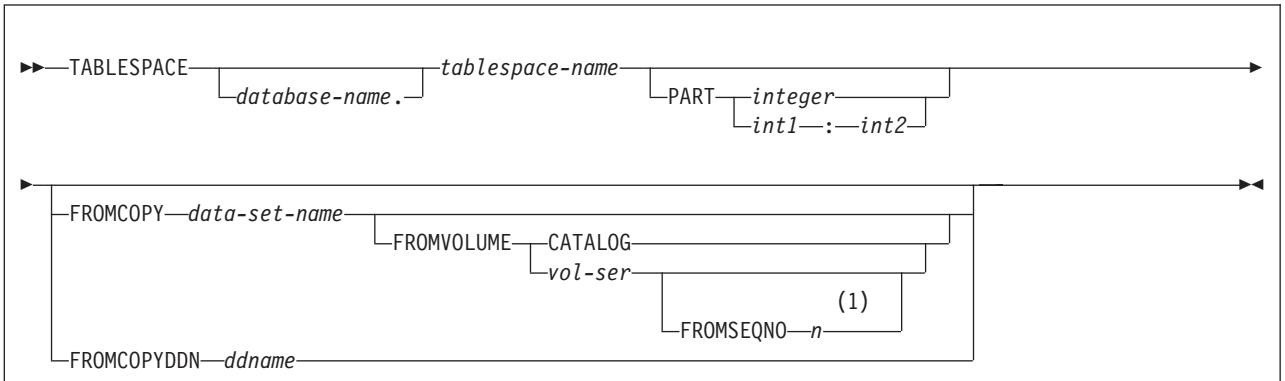
Syntax and options of the UNLOAD control statement

The utility control statement defines the function that the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name of the data set that contains the utility control statement.

Syntax diagram



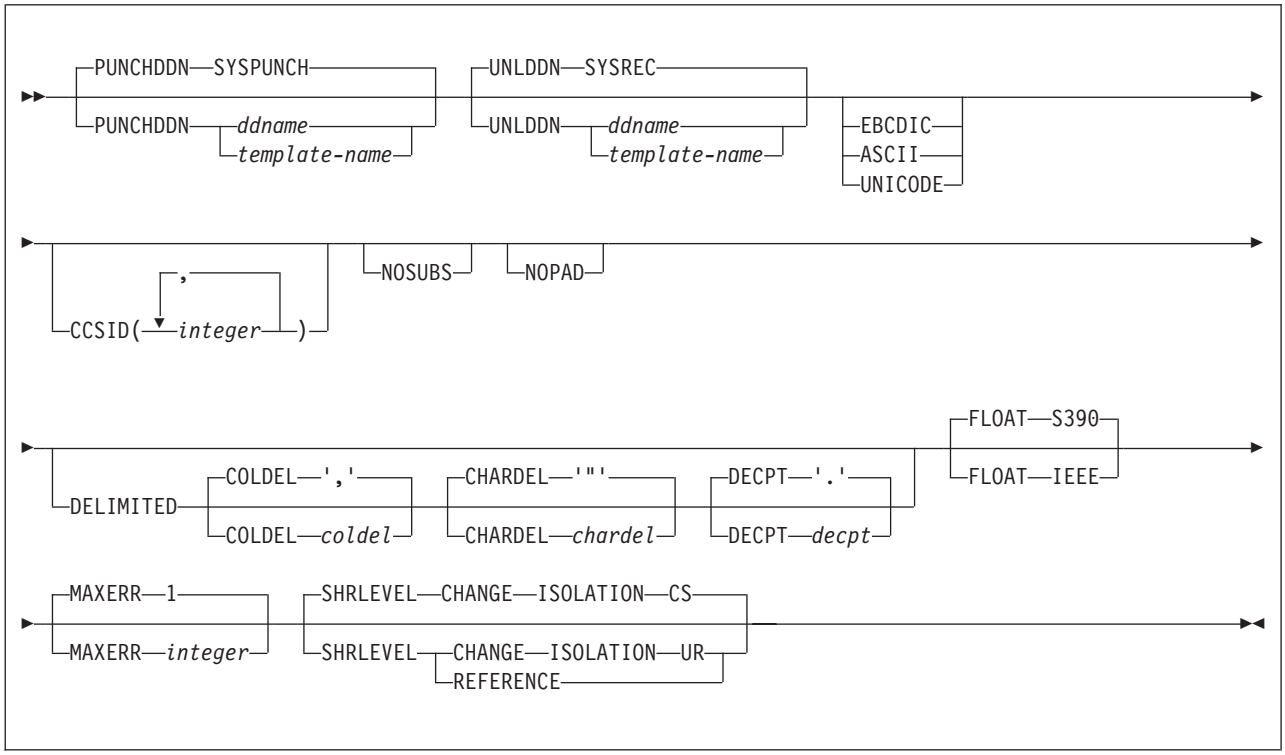
source-spec:



Notes:

1 The FROMSEQNO option is required if you are unloading an image copy from a tape data set that is not cataloged.

unload-spec:



FROM-TABLE-spec:

For the syntax diagram and the option descriptions of the FROM TABLE specification, see "FROM-TABLE-spec " on page 623.

Option descriptions

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

DATA Identifies the data that is to be selected for unloading with *table-name* in the *from-table-spec*. The DATA keyword is mutually exclusive with TABLESPACE, PART, and LIST keywords.

When you specify the DATA keyword, or you omit either the TABLESPACE or the LIST keyword, you must also specify at least one FROM TABLE clause.

TABLESPACE
Specifies the table space (and, optionally, the database to which it belongs) from which the data is to be unloaded.

database-name
The name of the database to which the table space belongs. The name cannot be DSNDB01 or DSNDB07. The **default** is **DSNDB04**.

tablespace-name
The name of the table space from which the data is to be unloaded. The specified table space must not be a LOB table space.

PART
Identifies a partition or a range of partitions from which the data is to

be unloaded. This keyword applies only if the specified table space is partitioned. You cannot specify PART with LIST. The maximum is 4096.

integer
Designates a single partition. *integer* must identify an existing partition number within the table space.

int1:int2
Designates a range of partitions from *int1* to *int2*. *int1* must be a positive integer that is less than the highest partition number within the table space. *int2* must be an integer that is greater than *int1* and less than or equal to the highest partition number.

If no PART keyword is specified in an UNLOAD control statement, the data from the entire table space is unloaded into a single unload data set.

FROMCOPY *data-set-name*
Indicates that data is to be unloaded from an image copy data set. When you specify FROMCOPY, the UNLOAD utility processes only the specified image copy data set. Alternatively, you can use the FROMCOPYDDN keyword where multiple image copy data sets can be concatenated under a single DD name.

data-set-name
Specifies the name of a single image copy data set.

The image copy data set that you specify on the FROMCOPY keyword must be created by one of the following utilities:
- COPY
- COPYTOCOPY
- LOAD inline image copy
- MERGECOPY
- REORG TABLESPACE inline image copy
- DSN1COPY

If the specified image copy data set is a full image copy, either compressed or uncompressed records can be unloaded.

If the default of SYSTEMPAGES was used for the incremental image copy, either compressed or uncompressed records can be unloaded.

If the specified image copy data set is an incremental image copy or a copy of a partition or partitions, you can unload compressed records only when the same data set contains the dictionary pages for decompression. If an image copy data set contains a compressed row and a dictionary is not available, DB2 issues an error message. See "MAXERR" on page 622 for more information about specifying error-tolerance conditions.

When you specify FROMCOPY or FROMCOPYDDN, you can also specify selection criteria with either PART, FROM TABLE, or both, to qualify tables and rows that are to be unloaded.

FROMVOLUME
Identifies the volume where the image copy data set resides.

CATALOG
Indicates that the data set is cataloged. Use this option only for an image copy that was created as a cataloged data set (which means that its volume serial is not recorded in SYSIBM.SYSCOPY).

vol-ser
Identifies the data set by an alphanumeric volume serial identifier of its first volume. Use this option only for an image copy that was created as a non-cataloged data set. To specify a data set that is stored on multiple tape volumes, identify the first *vol-ser* in the SYSCOPY record.

FROMSEQNO *n*
Identifies the image copy data set by its file sequence number. The FROMSEQNO option is required if you are unloading an image copy from a tape data set that is not cataloged.

n Specifies the file sequence number.

FROMCOPYDDN *ddname*
Indicates that data is to be unloaded from one or more image copy data sets that are associated with the specified *ddname*. Multiple image copy data sets (primarily for the copy of pieces) can be concatenated under a single DD name.

ddname
Identifies a DD name with which one or more image copy data sets are associated.

LIST *listdef-name*
Identifies the name of a list of objects that are defined by a LISTDEF utility control statement. The list can include table spaces, index spaces, databases, a tables, an index, and partitions. The list cannot include index spaces, LOB table spaces, and directory objects. You cannot use the LIST option to specify image copy data sets.

When you specify the LIST option, the referenced LISTDEF identifies:
- The table spaces from which the data is to be unloaded. You can use the pattern-matching feature of LISTDEF.
- The partitions (if a table space is partitioned) from which the data is to be unloaded (defined by the INCLUDE, EXCLUDE, and PARTLEVEL keywords in the LISTDEF statement).

The UNLOAD utility associates a single table space with one output data set, except when partition-parallelism is activated. When you use the LIST option with a LISTDEF that represents multiple table spaces, you must also define a data set TEMPLATE that corresponds to all of the table spaces and specify the *template-name* in the UNLDDN option.

If you want to generate the LOAD statements, you must define another TEMPLATE for the PUNCHDDN data set that is similar to UNLDDN. DB2 then generates a LOAD statement for each table space.

PUNCHDDN
Specifies the DD name for a data set or a template name that defines one or more data set names that are to receive the LOAD utility control statements that the UNLOAD utility generates.

ddname
Specifies the DD name. The **default** is **SYSPUNCH**.

template-name
Identifies the name of a data set template that is defined by a TEMPLATE utility control statement.

If the specified name is defined both as a DD name (in the JCL) and as a template name (in a TEMPLATE statement), it is treated as the DD name.

When you run the UNLOAD utility for multiple table spaces and you want to generate corresponding LOAD statements, you must have multiple output data sets that correspond to the table spaces so that DB2 retains all of the generated LOAD statements. In this case, you must specify an appropriate template name to PUNCHDDN. If you omit the PUNCHDDN specification, the LOAD statements are not generated.

If the partition variable (&PART or &PA) is included in a TEMPLATE for PUNCHDDN, DB2 replaces the &PART or &PA variable with the lowest partition number in the list of partitions to be unloaded. The partition number is in the form *nnnn*.

UNLDDN

Specifies the DD name for a data set or a template name that defines one or more data set names into which the data is to be unloaded.

ddname
Specifies the DD name. The **default** is **SYSREC**.

template-name
Identifies the name of a data set template that is defined by a TEMPLATE utility control statement.

If the specified name is defined both as a DD name (in the JCL) and as a template name (in a TEMPLATE statement), it is treated as the DD name.

When you run the UNLOAD utility for a partitioned table space, the selected partitions are unloaded in parallel if the following conditions are true:

1. You specify a template name for UNLDDN.
2. The template data set name contains the partition as a variable (&PART. or &PA.) without substring notation. This template name is expanded into multiple data sets that correspond to the selected partitions.
3. The TEMPLATE control statement does not contain all of the following options:
   - STACK(YES)
   - UNIT(TAPE)
   - An UNCNT value that is less than or equal to one.

If conditions 1 and 2 are true, but condition 3 is false, partition parallelism is not activated and all output data sets are stacked on one tape.

If condition 2 is false because &PA(s,l). substring syntax is used, the DSN may not be unique for all partitions and parallel UNLOAD cannot be performed. Therefore the &PA. variable is set to zero and a single UNLDDN data set is used for all partitions. This action may cause duplicate data set errors on subsequent UNLOAD jobs for other partitions of the same table space.

Similarly, when you run the UNLOAD utility for multiple table spaces, the output records are placed in data sets that correspond to the respective table spaces. Therefore the output data sets must be physically distinctive, and you must specify an appropriate template name to UNLDDN. If you omit the UNLDDN specification, the SYSREC DD name is not used, and an error occurs.

If the partition variable (&PART. or &PA.) is included in the TEMPLATE DSN statement when partition parallelism is not applicable (when the source is a non-partitioned table space or an image copy), the variable is replaced by '00000' in the actual data set name. In this case, warning message DSNU1252I is issued, and the UNLOAD utility issues return code 4.

EBCDIC
Specifies that all output data of the character type is to be in EBCDIC. If a different encoding scheme is used for the source data, the data (except for bit strings) is converted into EBCDIC.

If you do not specify EBCDIC, ASCII, UNICODE, or CCSID, the encoding scheme of the source data is preserved.

See the description of the CCSID option for this utility.

ASCII Specifies that all output data of the character type is to be in ASCII. If a different encoding scheme is used for the source data, the data (except for bit strings) is converted into ASCII.

If you do not specify EBCDIC, ASCII, UNICODE, or CCSID, the encoding scheme of the source data is preserved.

See the description of the CCSID option for this utility.

UNICODE
Specifies that all output data of the character type (except for bit strings) is to be in Unicode. If a different encoding scheme is used for the source data, the data is converted into Unicode.

If you do not specify EBCDIC, ASCII, UNICODE, or CCSID, the encoding scheme of the source data is preserved.

See the description of the CCSID option of this utility.

CCSID*(integer1,integer2,integer3)*
Specifies up to three coded character set identifiers (CCSIDs) that are to be used for the data of character type in the output records, including data that is unloaded in the external character formats.

integer1 specifies the CCSID for SBCS data. *integer2* specifies the CCSID for mixed data. *integer3* specifies the CCSID for DBCS data. This option is not applied to data with a subtype of BIT.

| If you specify both FORMAT DELIMITED and UNICODE, all output data
| is in CCSID 1208, UTF-8; any other specified CCSID is ignored.

The following specifications are also valid:

CCSID(*integer1***)**
Indicates that only an SBCS CCSID is specified.

CCSID(*integer1,integer2***)**
Indicates that an SBCS CCSID and a mixed CCSID are specified.

integer
Specifies either a valid CCSID or 0.

If you specify a value of 0 for one of the arguments or omit a value, the encoding scheme that is specified by EBCDIC, ASCII, or UNICODE is assumed for the corresponding data type (SBCS, MIXED, or DBCS). If you do not specify EBCDIC, ASCII, or UNICODE:

- If the source data is of character type, the original encoding scheme is preserved.
- For character strings that are converted from numeric, date, time, or timestamp data, the default encoding scheme of the table is used. For more information, see the CCSID option of the CREATE TABLE statement in Chapter 5 of *DB2 SQL Reference*.

When a CCSID conversion is requested, CCSID character substitutions can occur in the output string. Use the NOSUBS option to prevent possible character substitutions during CCSID conversion.

NOSUBS
Specifies that CCSID code substitution is not to be performed during unload processing.

When a string is converted from one CCSID to another (including EBCDIC, ASCII, and Unicode), a substitution character is sometimes placed in the output string. For example, this substitution occurs when a character (referred to as a code point) that exists in the source CCSID does not exist in the target CCSID. You can use the NOSUBS keyword to prevent the UNLOAD utility from allowing this substitution.

If you specify the NOSUBS keyword and character substitution is attempted while data is being unloaded, this action is treated as a conversion error. The record with the error is not unloaded, and the process continues until the total error count reaches the number that is specified by MAXERR.

NOPAD
Specifies that the variable-length columns in the unloaded records are to occupy the actual data length without additional padding. As a result, the unloaded or discarded records might have varying lengths.

When you do not specify NOPAD:

- Default UNLOAD processing pads variable-length columns in the unloaded records to their maximum length, and the unloaded records have the same length for each table.
- The padded data fields are preceded by the length fields that indicate the size of the actual data without the padding.
- When the output records are reloaded with the LOAD utility, padded data fields are treated as varying-length data.

If you specify DELIMITED, the NOPAD option is the default for variable-length columns. For fixed-length columns, the normal padding rules apply.

Although LOAD processes records with variable-length columns that are unloaded or discarded by using the NOPAD option, these records cannot be processed by applications that process only fields in fixed positions. For example, the LOAD statement that is generated for the EMP sample table would look similar to the LOAD statement in Figure 77 on page 437.

DELIMITED
Specifies that the output data file is in a delimited format. When data is in a delimited format, all fields in the output data set are character strings or external numeric values. In addition, each column in a delimited file is separated from the next column by a column delimiter character.

For each of the delimiter types that you can specify, you must ensure that the delimiter character is specified in the code page of the target data. The

delimiter character can be specified as either a character or hex constant. For example, to specify # as the delimiter, you can specify either COLDEL '#' or COLDEL X'23'. If the utility statement is coded in a character type that is different from the output file, such as a utility statement that is coded in EBCDIC and output data that is in Unicode, specify the delimiter character in the utility statement as a hex constant, or the result is unpredictable.

You cannot specify the same character for more than one type of delimiter (COLDEL, CHARDEL, and DECPT).

If you specify the FORMAT DELIMITED option, you cannot specify HEADER CONST or use any of the multiple FROM TABLE statements. Also, UNLOAD ignores any specified POSITION statements within the UNLOAD utility control statement.

For delimited output, UNLOAD does not add trailing padded blanks to variable-length columns, even if you do not specify the NOPAD option. For fixed-length columns, the normal padding rules apply. For example, if a VARCHAR(10) field contains `ABC`, UNLOAD DELIMITED unloads the field as `"ABC"`. However, for a CHAR(10) field that contains `ABC`, UNLOAD DELIMITED unloads it as `"ABC       "`. For information about using delimited output and delimiter restrictions, see "Unloading delimited files" on page 656. For more information about delimited files see Appendix F, "Delimited file format," on page 899.

COLDEL
Specifies the column delimiter that is used in the output file. The default is a comma (,). For ASCII and UTF-8 data this is X'2C', and for EBCDIC data it is a X'6B'.

CHARDEL
Specifies the character string delimiter that is used in the output file. The default is a double quotation mark ("). For ASCII and UTF-8 data this is X'22', and for EBCDIC data it is X'7F'.

The UNLOAD utility adds the CHARDEL character before and after every character string. To delimit character strings that contain the character string delimiter, the UNLOAD utility repeats the character string delimiter where it used in the character string. The LOAD utility then interprets any pair of character delimiters that are found between the enclosing character delimiters as a single character. For example, the phrase `what a "nice warm" day` is unloaded as `"what a ""nice warm"" day"`, and LOAD interprets it as `what a "nice warm" day`. The UNLOAD utility recognizes these character pairs for only CHAR, VARCHAR, and CLOB fields.

DECPT
Specifies the decimal point character that is used in the output file. The default is a period (.).

The default decimal point character is a period in a delimited file, X'2E' in an ASCII data file, and X'4B' in an EBCDIC data file.

FLOAT
Specifies the output format of the numeric floating-point data. This option applies to the binary output format only.

S390
Indicates that the binary floating point data is written to the output records in the S/390 internal format (also known as the hexadecimal floating point, or HFP).

The **default** is **FLOAT S390**.

IEEE
Indicates that the binary floating-point data is written to the output records in the IEEE format (also known as the binary floating point, or BFP).

MAXERR *integer*
Specifies the maximum number of records in error that are to be allowed; the unloading process terminates when this value is reached.

integer
Specifies the number of records in error that are allowed. When the error count reaches this number, the UNLOAD utility issues message DSNU1219 and terminates with return code 8.

The **default** is **1**, which indicates that UNLOAD stops when the first error is encountered. If you specify 0 or any negative number, execution continues regardless of the number of records that are in error.

If multiple table spaces are being processed, the number of records in error is counted for each table space. If the LIST option is used, you can add OPTION utility control statement (EVENT option with ITEMERROR) before the UNLOAD statement to specify that the table space in error is to be skipped and the subsequent table spaces are to be processed.

SHRLEVEL
Specifies whether other processes can access or update the table space or partitions while the data is being unloaded.

UNLOAD ignores the SHRLEVEL specification when the source object is an image copy data set.

The **default** is **SHRLEVEL CHANGE ISOLATION CS**.

CHANGE
Specifies that rows can be read, inserted, updated, and deleted from the table space or partition while the data is being unloaded.

ISOLATION
Specifies the isolation level with SHRLEVEL CHANGE.

CS
Indicates that the UNLOAD utility is to read rows in cursor stability mode. With CS, the UNLOAD utility assumes CURRENTDATA(NO).

UR
Indicates that uncommitted rows, if they exist, are to be unloaded. The unload operation is performed with minimal interference from the other DB2 operations that are applied to the objects from which the data is being unloaded.

REFERENCE
Specifies that during the unload operation, rows of the tables can be read, but cannot be inserted, updated, nor deleted by other DB2 threads.

When you specify SHRLEVEL REFERENCE, the UNLOAD utility drains writers on the table space from which the data is to be unloaded. When data is unloaded from multiple partitions, the drain lock is obtained for all of the selected partitions in the UTILINIT phase.

FROM-TABLE-spec

More than one table or partition for each table space can be unloaded with a single invocation of the UNLOAD utility. One FROM TABLE statement for each table that is to be unloaded is required to identify:

- A table name from which the rows are to be unloaded
- A field to identify the table that is associated with the rows that are to be unloaded from the table by using the HEADER option
- Sampling options for the table rows
- A list of field specifications for the table that is to be used to select columns that are to be unloaded
- Selection conditions, specified in the WHEN clause, that are to be used to qualify rows that are to be unloaded from the table

All tables that are specified by FROM TABLE statements must belong to the same table space. If rows from specific tables are to be unloaded, a FROM TABLE clause must be specified for each source table. If you do not specify a FROM TABLE clause for a table space, all the rows of the table space are unloaded.

Use a list of field specifications to specify the following characteristics:

- Column selection. Specifies the column names of a table that is to be unloaded. If a list of field specifications is given, only the listed columns are unloaded.
- Column ordering. Specifies the order of fields that are to be placed in the output records. If a list of field specifications is given, data of the listed columns is unloaded in the order of listed column names.
- Output field attributes and format. Specifies the data type, length, and format of the data in the output records.

If you omit a list of field specifications, all columns of the source table are unloaded in the defined column order for the table. The default output field types that correspond to the data types of the columns are used.

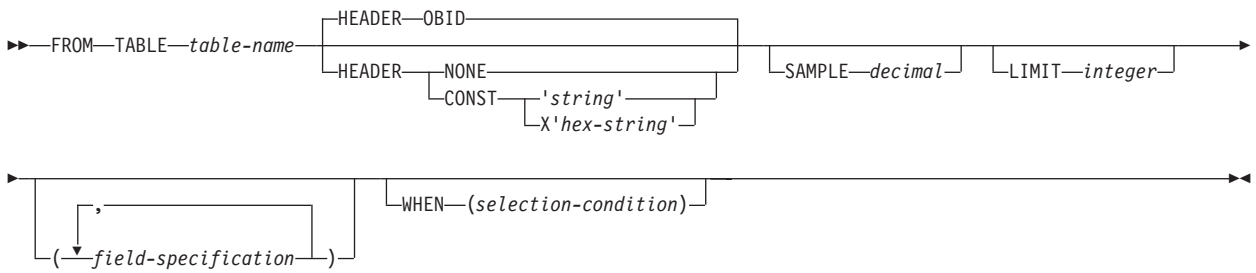
In a FROM TABLE clause, you can use parentheses in only two situations: to enclose the entire field selection list, and in a WHEN selection clause. This usage avoids potential conflict between the keywords and field-names that are used in the field selection list. A valid sample of a FROM TABLE clause specification follows:

```
UNLOAD ...
  FROM TABLE tablename SAMPLE x (c1,c2) WHEN (c3>0)
```

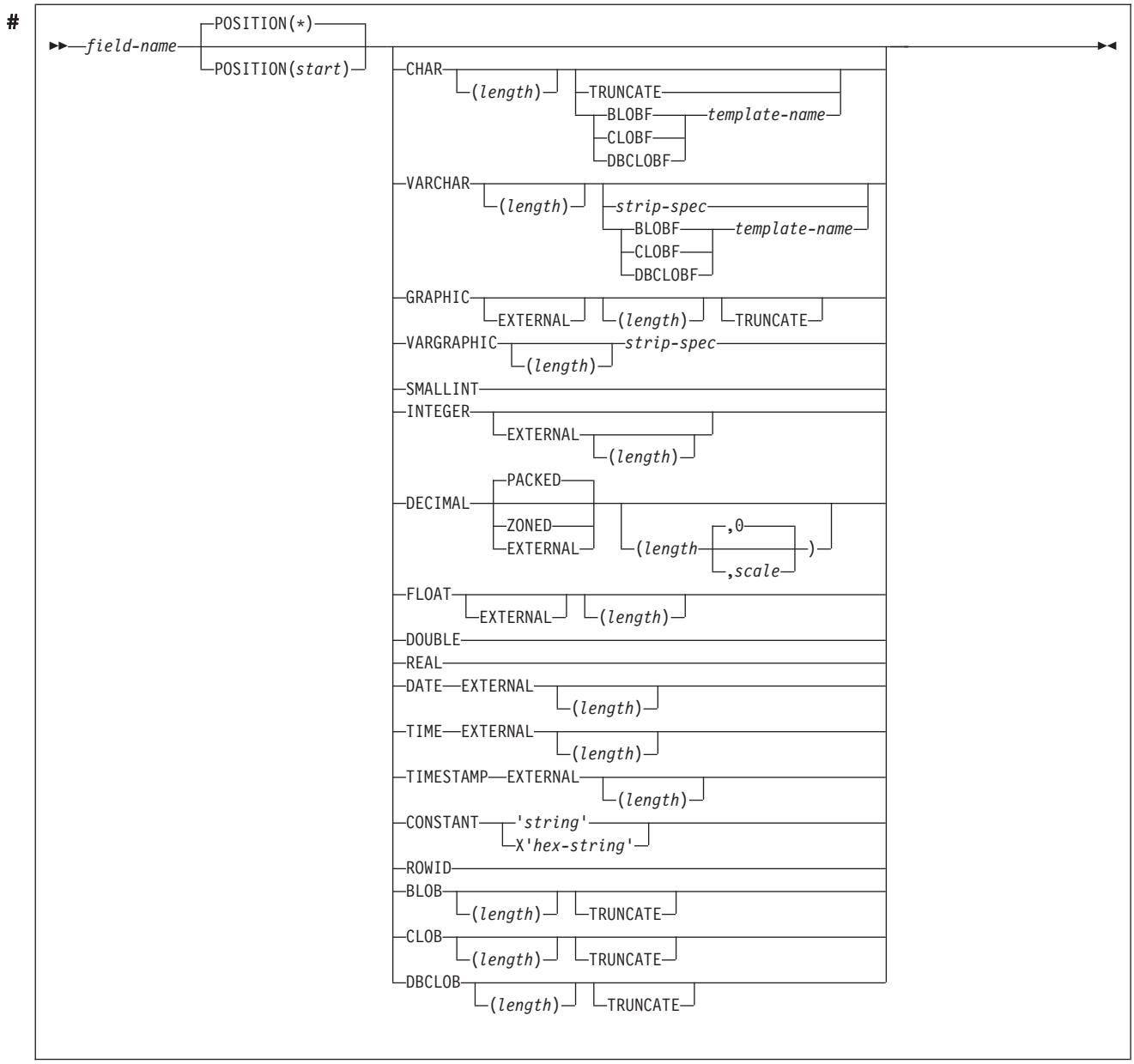
You cannot specify FROM TABLE if the LIST option is already specified.

FROM-TABLE-spec:

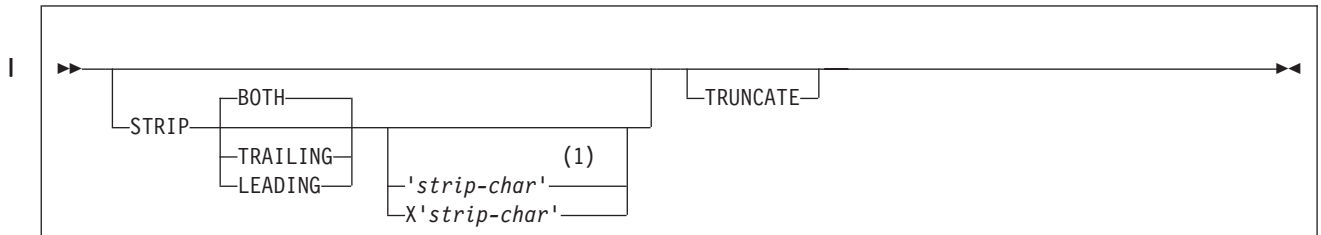
FROM-TABLE-spec



field-specification:



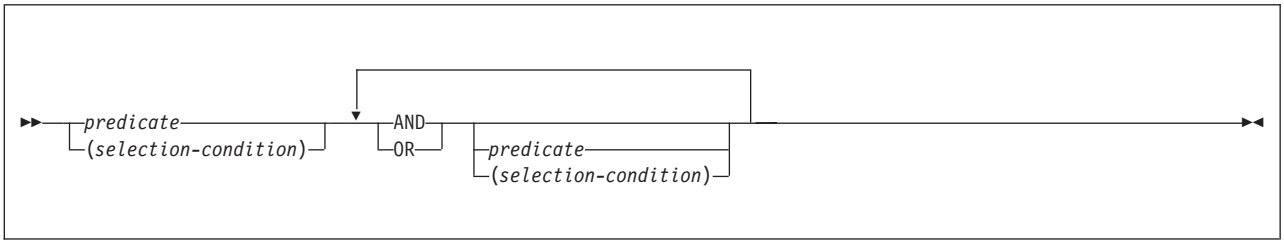
strip spec:



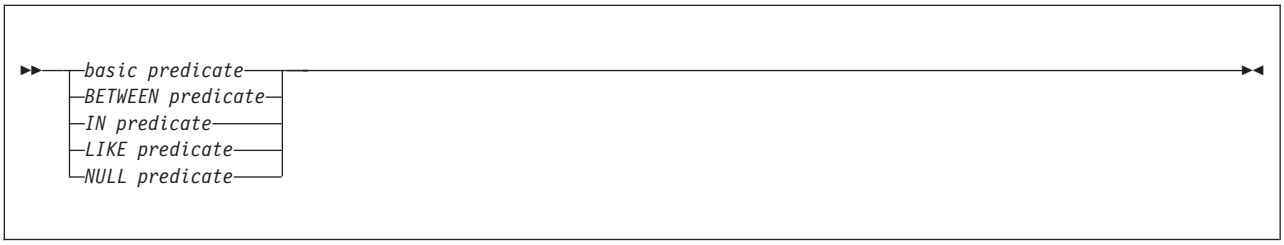
Notes:

1 If you specify VARGRAPHIC, you cannot specify 'strip-char'. You can specify only X'strip-char'.

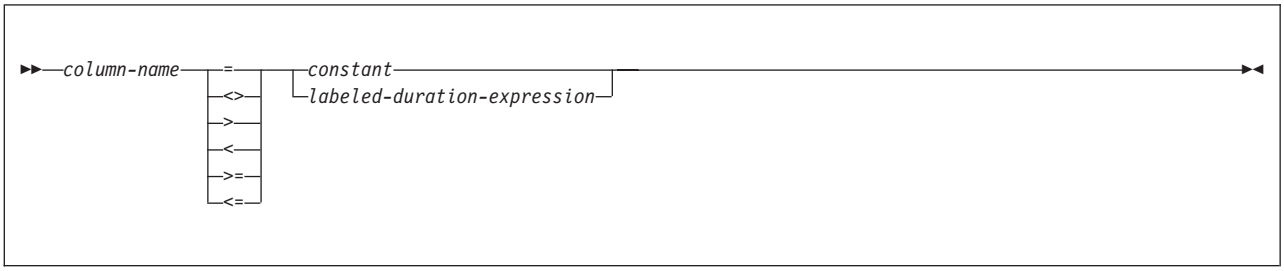
selection condition:



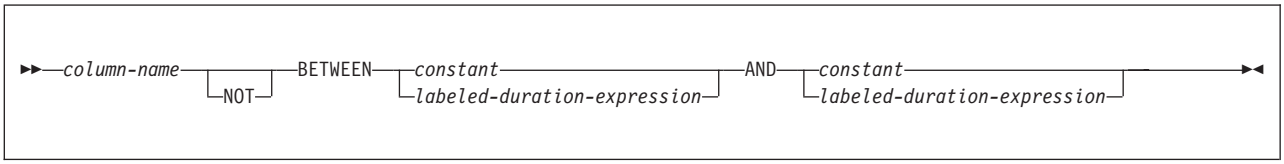
predicate:



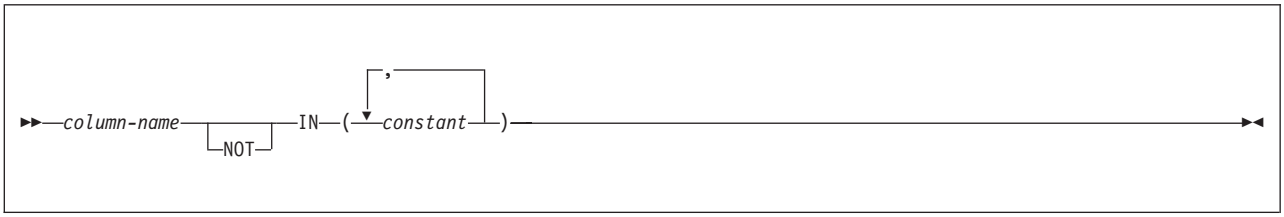
basic predicate:



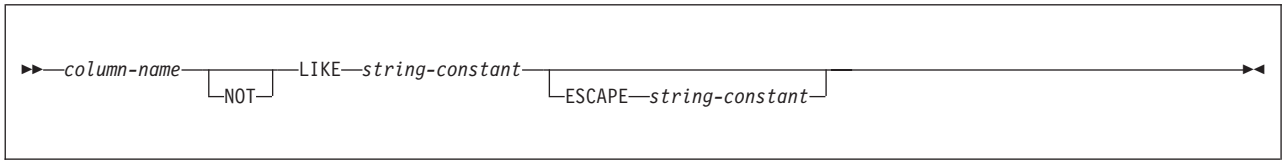
BETWEEN predicate:



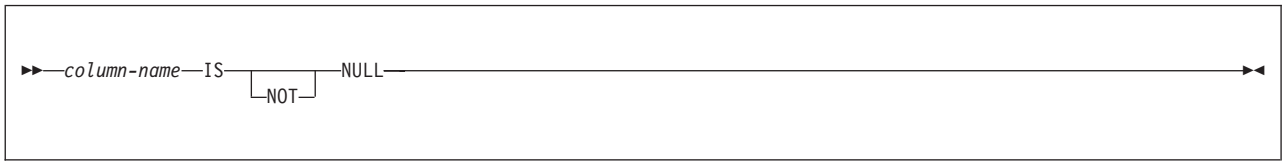
IN predicate:



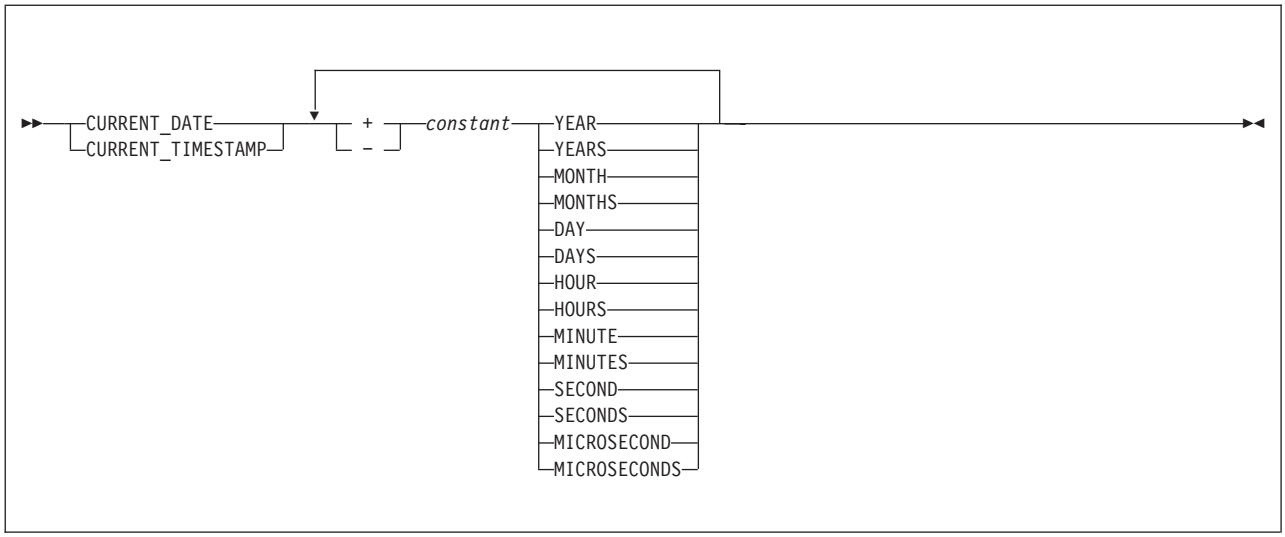
LIKE predicate:



NULL predicate:



labeled-duration-expression:



Option descriptions for FROM TABLE

"Control statement coding rules" on page 16 provides general information about specifying options for DB2 utilities.

table-name

Identifies a DB2 table from which the rows are to be unloaded and to which the options in the FROM TABLE clause are to be applied.

If the table name is not qualified by an authorization ID, the authorization ID of the invoker of the utility job step is used as the qualifier of the table name. Enclose the table name in quotation marks if the name contains a blank.

HEADER

Specifies a constant header field, at the beginning of the output records, that can be used to associate an output record with the table from which it was unloaded.

If you specify a header field, it is used as the field selection criterion of the WHEN clause (a part of the INTO-TABLE specification) in the LOAD statement that is generated.

OBID
Specifies that the object identifier (OBID) for the table (a 2-byte binary value) is to be placed in the first 2 bytes of the output records that are unloaded from the table.

If you omit the HEADER option, **HEADER OBID** is the default, except for delimited files.

With HEADER OBID, the first 2 bytes of the output record cannot be used by the unloaded data. For example, consider the following UNLOAD statement:

```
UNLOAD ...
  FROM TABLE table-name HEADER OBID ...
```

The preceding UNLOAD statement generates a LOAD statement that is similar to the following example:

```
LOAD ...
  INTO TABLE table-name WHEN (1:2)=X'hh' ...
```

In this example, X'hh' is the hexadecimal notation of the OBID of table *table-name*.

NONE
Indicates that no record header field is to be created. HEADER NONE is the default value for a delimited file.

If HEADER NONE is specified in a FROM TABLE clause, the corresponding INTO TABLE clause in the generated LOAD statement does not have a WHEN specification. Therefore, if rows from multiple tables are unloaded and HEADER NONE is specified in one or more FROM TABLE clauses, rows that are unloaded from those tables are not able to be reloaded until you edit the generated LOAD statement. If you use the generated statement directly with the LOAD utility, the results might be unpredictable.

CONST
Specifies that a constant string is to be used as the record header. The given string operand determines the length of the header field. The string value must be enclosed by a pair of single quote characters.

For example, consider the following UNLOAD statement:

```
UNLOAD ...
  FROM TABLE table-name HEADER CONST 'abc' ...
```

The preceding UNLOAD statement generates a LOAD statement that is similar to the following example:

```
LOAD ...
  INTO TABLE table-name WHEN (1:3)='abc' ...
```

In this example, the given string is assumed to be in SBCS EBCDIC format. The output string of the HEADER field is in the specified or the default encoding scheme. If the encoding scheme that is used for output is not EBCDIC, the SBCS CCSID conversion is applied to the given string before it is placed in the output records. If the output

SBCS encoding scheme is not EBCDIC, the WHEN condition in the generated LOAD statement contains a hexadecimal string.

You can also use the hexadecimal form, X'*hex-string*', to represent a string constant. If you want to specify a CONST string value in an encoding scheme other than SBCS EBCDIC, use the hexadecimal form. No CCSID conversion is performed if the hexadecimal form is used.

SAMPLE *decimal*

Indicates that only sampled rows of the table are to be unloaded. If selection conditions are specified by a WHEN clause within the same FROM TABLE clause, sampling is applied to the rows that are qualified by the WHEN selection conditions.

decimal

Specifies the percentage of the rows that are to be sampled in the decimal format. The precision is *ddd.dddd*, and the valid range is *0 <= decimal <= 100*.

If the number of rows to which the sampling is to be applied is *N*:

- *decimal* × *N* / *100* rows are unloaded. (The fraction might be rounded to the nearest whole number.)
- If *decimal* > *0* and *N* > *0*, at least one row is unloaded.
- If *decimal* = *100*, all rows from the table are unloaded.
- If the given *decimal* = *0* or *N* = *0*, no row is unloaded from the table.

The sampling is applied for each individual table. If the rows from multiple tables are unloaded with sampling enabled, the referential integrity between the tables might be lost.

LIMIT *integer*

Specifies the maximum number of rows that are to be unloaded from a table. If the number of unloaded rows reaches the specified limit, message DSNU1202I is issued for the table, and no more rows are unloaded from the table. The process continues to unload qualified rows from the other tables.

When partition parallelism is activated, the LIMIT option is applied to each partition instead of to the entire table.

integer

Indicates the maximum number of rows that are to be unloaded from a table. If the specified number is less than or equal to zero, no row is unloaded from the table.

Like the SAMPLE option, if multiple tables are unloaded with the LIMIT option, the referential integrity between the tables might be lost.

field-name

Identifies a column name that must exist in the source table.

POSITION*(start)*

Specifies the field position in the output record. You can specify

the position parameter as follows:

* An asterisk, indicating that the field starts at the first byte after the last position of the previous field.

start A positive integer that indicates the start column of the data field.

The **default** is **POSITION(*)**.

The first column (byte position) of an output record corresponds to POSITION(1). If you specify HEADER NONE in the FROM TABLE clause, the item that is specified by the HEADER option is placed at the beginning of all the records that are unloaded from the table. You must account for the space for the record header:

- HEADER OBID (the default case): 2 bytes from position 1.
- HEADER CONST 'string' or X'*hex-string*' case: The length of the given string from position 1.

If the source table column can be null, the utility places a NULL indicator byte at the beginning of the data field in the output record. For BLOBF, CLOBF, or DBCLOBF columns, null values are indicated by a byte at the beginning of the file name. The *start* parameter (or *) points to the position of the NULL indicator byte. In the generated LOAD statement, *start* is shifted by 1 byte to the right (as *start+1*) so that, in the LOAD statement, the start parameter of the POSITION option points to the next byte past the NULL indicator byte.

For a varying-length field, a length field precedes the actual data field (after the NULL indicator byte, if applicable). For BLOBF, CLOBF, or DBCLOBF columns, the length of the file name is indicated by two bytes at the beginning of the file name.If the value cannot be null, the *start* parameter (or *) points to the first byte of the length field. The size of the length field is either 4 bytes (BLOB, CLOB, or DBCLOB) or 2 bytes (VARCHAR or VARGRAPHIC).

When you explicitly specify the output field positions by using *start* parameters (or using the * format) of the POSITION option, you must consider the following items as a part of the output field:

- For a field whose value can be null, a space for the NULL indicator byte
- For varying-length data, a space for the length field (either 2 bytes or 4 bytes)

"Determining the layout of output fields" on page 653 illustrates the field layout in conjunction with the POSITION option, NULL indicator byte, the length field for a varying-length field, the *length* parameter, and the actual data length.

The POSITION option is useful when the output fields must be placed at desired positions in the output records. The use of the POSITION parameters, however, can restrict the size of the output data fields. Use care when explicitly specifying *start* parameters for nullable and varying-length fields. The TRUNCATE option might be required, if applicable, to fit a data item in a shorter space in an output record.

If you omit the POSITION option for the first field, the field starts from position 1 if HEADER NONE is specified. Otherwise, the field starts from the next byte position past the record header field. If POSITION is omitted for a subsequent field, the field is placed next to the last position of the previous field without any gap.

If NOPAD is specified and POSITION parameters are given for certain fields, the effect of the NOPAD option might be lost because the fields with *start* parameters (other than the default *) always start at the fixed positions in the output records.

The POSITION option is ignored for delimited output files.

CHAR

Indicates that the output field is a character type with fixed length. You can use CHARACTER in place of CHAR. If the source table column can be null, a NULL indicator byte is placed at the beginning of the output field for a non-delimited output file.

If you specify the EBCDIC, ASCII, UNICODE, or CCSID options, the output data that corresponds to the specified option, is encoded in the CCSID, depending on the subtype of the source data (SBCS or MIXED). If the subtype is BIT, no conversion is applied. If the output field is a file name, the output data in the file is in the SBCS CCSID of the specified encoding scheme.

For BLOBF, CLOBF, and DBCLOBF, if the file does not exist, DB2 creates it using the attributes of the specified template. If the template specifies a DSNTYPE of LIBRARY or PDS, DB2 appends (&UNIQ.) to the *name-expression* of the template. If the template specifies a DSNTYPE of HFS, DB2 appends /UNIQ. to the *name-expression* of the template.

Restriction: BLOBF, CLOBF, and DBCLOBF are not supported when unloading from an image copy.

(length)

Specifies the size of the output data in bytes.

If the *length* parameter is omitted, the default is the maximum length that is defined on the source table column. When the *length* parameter is specified:

- If the *length* is less than the size of the table column, the data is truncated to the *length* if the TRUNCATE keyword is present; otherwise, a conversion error occurs.
- If the *length* is larger than the size of the table column, the output field is padded by the default pad characters to the specified length.

BLOBF

Specifies that the output field is to contain the name of the file to which the BLOB is to be unloaded without CCSID conversion.

CLOBF

Specifies that the output field is to contain the name of the file to which the CLOB is to be unloaded with any required CCSID conversion.

DBLOBF

Specifies that the output field is to contain the name of the file to which the DBLOB is to be unloaded with any required CCSID conversion.

TRUNCATE

Indicates that a character string (encoded for output) is to be truncated from the right, if the data does not fit in the available space for the field in the output record. Truncation occurs at the character boundary. See “Specifying TRUNCATE and STRIP options for output data” on page 658 for the truncation rules that are used in the UNLOAD utility. Without TRUNCATE, an error occurs when the output field size is too small for the data.

VARCHAR

Specifies that the output field type is character of varying length. A 2-byte binary field indicating the length of data in bytes is prepended to the data

field. If the table column can be null, a NULL indicator byte is placed before this length field for a non-delimited output file.

If you specify the EBCDIC, ASCII, UNICODE, or CCSID options, the output data is encoded in the CCSID corresponding to the specified option, depending on the subtype of the source data (SBCS or MIXED). If the subtype is BIT, no conversion is applied. If the output field is a file name, the output data in the file is in the SBCS CCSID of the specified encoding scheme.

For BLOBF, CLOBF, and DBCLOBF, if the file does not exist, DB2 creates it using the attributes of the specified template. If the template specifies a DSNTYPE of LIBRARY or PDS, DB2 appends (&UNIQ.) to the *name-expression* of the template. If the template specifies a DSNTYPE of HFS, DB2 appends /UNIQ. to the *name-expression* of the template.

Restriction: BLOBF, CLOBF, and DBCLOBF are not supported when unloading from an image copy.

(length)
Specifies the maximum length of the actual data field in bytes. If you also specify NOPAD, it indicates the maximum allowable space for the data in the output records; otherwise, the space of the specified length is reserved for the data.

If the length parameter is omitted, the default is the smaller of 255 and the maximum length that is defined on the source table column.

BLOBF
Specifies that the output field is to contain the name of the file to which the BLOB is to be unloaded without CCSID conversion.

CLOBF
Specifies that the output field is to contain the name of the file to which the CLOB is to be unloaded with any required CCSID conversion.

DBLOBF
Specifies that the output field is to contain the name of the file to which the DBLOB is to be unloaded with any required CCSID conversion.

STRIP
Specifies that UNLOAD is to remove blanks (the default) or the specified characters from the beginning, the end, or both ends of the data. UNLOAD adjusts the VARCHAR length field (for the output field) to the length of the stripped data.

The STRIP option is applicable if the subtype of the source data is BIT. In this case, no CCSID conversion is performed on the specified strip character (even if it is given in the form *'strip-char'*).

The effect of the STRIP option is the same as the SQL STRIP scalar function. For details, see Chapter 5 of *DB2 SQL Reference*.

BOTH
Indicates that UNLOAD is to remove occurrences of blank or the specified strip character from the beginning and end of the data. The **default** is **BOTH**.

TRAILING
Indicates that UNLOAD is to remove occurrences of blank or the specified strip character from the end of the data.

LEADING
Indicates that UNLOAD is to remove occurrences of blank or the specified strip character from the beginning of the data.

'strip-char'
Specifies a single-byte character that is to be stripped. Specify this character value in EBCDIC. Depending on the output encoding scheme, UNLOAD applies SBCS CCSID conversion to the *strip-char* value before it is used in the strip operation. If you want to specify a *strip-char* value in an encoding scheme other than EBCDIC, use the hexadecimal form. UNLOAD does not perform CCSID conversion if the hexadecimal form is used.

X'strip-char'
Specifies a single-byte character that is to be stripped. It can be specified in the hexadecimal form, X'*hex-string*', where *hex-string* is two hexadecimal characters that represent a single SBCS character. If the *strip-char* operand is omitted, the default is the blank character, which is coded as follows:
- X'40', for the EBCDIC-encoded output case
- X'20' for the ASCII-encoded output case
- X'20' the Unicode-encoded output case

The strip operation is applied after the character code conversion, if the output character encoding scheme is different from the one that is defined on the source data. Therefore, if a strip character is specified in the hexadecimal format, you must specify the character in the encoding scheme that is used for output.

TRUNCATE
Indicates that a character string (encoded for output) is to be truncated from the right, if the data does not fit in the available space for the field in the output records. Truncation occurs at a character boundary. See "Specifying TRUNCATE and STRIP options for output data" on page 658 for the truncation rules that are used in the UNLOAD utility. Without TRUNCATE, an error occurs when the output field size is too small for the data.

GRAPHIC
Specifies that the output field is of the fixed-length graphic type. If the table column can be null, a NULL indicator byte is placed before the actual data field for any non-delimited output file.

If the output is in EBCDIC, the shift-in and shift-out characters are not included at the beginning and at the end of the data.

(length)
Specifies the number of DBCS characters (the size of the output data in bytes is twice the given length). If the given *length* is larger than the source data length, the output field is padded with the default pad character.

TRUNCATE
Indicates that a graphic character string (encoded for output) is to be truncated from the right, if the data does not fit in the available space for the field in the output records. Truncation occurs at a character

(DBCS) boundary. Without TRUNCATE, an error occurs when the output field size is too small for the data.

GRAPHIC EXTERNAL
Specifies that the data is to be written in the output records as a fixed-length field of the graphic type with the external format; that is, the shift-out (SO) character is placed at the starting position, and the shift-in (SI) character is placed at the ending position. The byte count of the output field is always an even number.

GRAPHIC EXTERNAL is supported only in the EBCDIC output mode (by default or when the EBCDIC keyword is specified).

If the *start* parameter of the POSITION option is used to specify the output column position, it points to the (inserted) shift-out character at the beginning of the field. The shift-in character is placed at the next byte position past the last double-byte character of the data.

(length)
Specifies a number of DBCS characters, excluding the shift characters (as in the graphic type column definition that is used in a CREATE TABLE statement) nor the NULL indicator byte if the source column can be null. If the length parameter is omitted, the default output field size is the length that is defined on the corresponding table column, plus two bytes (shift-out and shift-in characters).

If the specified *length* is larger than the size of the data, the field is padded on the right with the default DBCS padding character.

TRUNCATE
Indicates that a graphic character string is to be truncated from the right by the DBCS characters, if the data does not fit in the available space for the field in the output records. Without TRUNCATE, an error occurs when the output field size is too small for the data. An error can also occur with the TRUNCATE option if the available space is less than 4 bytes (4 bytes is the minimum size for a GRAPHIC EXTERNAL field; shift-out character, one DBCS, and shift-in character); or fewer than 5 bytes if the field is can be null (the 4 bytes plus the NULL indicator byte).

VARGRAPHIC
Specifies that the output field is to be of the varying-length graphic type. A 2-byte binary length field is prepended to the actual data field. If the table column can be null, a NULL indicator byte is placed before this length field for any non-delimited output file.

(length)
Specifies the maximum length of the actual data field in the number of DBCS characters. If you also specify NOPAD, it indicates the maximum allowable space for the data in the output records; otherwise, the space of the specified length is reserved for the data.

If the length parameter is omitted, the default is the smaller of 127 and the maximum defined length of the source table column.

STRIP
Indicates that UNLOAD is to remove DBCS blanks (the default) or the specified characters from the unloaded data. UNLOAD adjusts the VARGRAPHIC length field (for the output field) to the length of the stripped data (the number of DBCS characters).

The effect of the STRIP option is the same as the SQL STRIP scalar function. For details, see Chapter 5 of *DB2 SQL Reference*.

BOTH
Indicates that UNLOAD is to remove occurrences of blank or the specified strip character from the beginning and end of the data. The **default** is **BOTH**.

TRAILING
Indicates that UNLOAD is to remove occurrences of blank or the specified strip character from the end of the data.

LEADING
Indicates that UNLOAD is to remove occurrences of blank or the specified strip character from the beginning of the data.

X'strip-char'
Specifies a DBCS character that is to be stripped in the hexadecimal format, **X**'*hhhh*', where *hhhh* is four hexadecimal characters that represent a DBCS character. If this operand is omitted, the default is a DBCS blank in the output encoding scheme (for example, X'4040' for the EBCDIC-encoded output or X'8140' for CCSID 301).

The strip operation is applied after the character code conversion, if the output character encoding scheme is different from the one that is defined on the source data. Therefore, if you specify a strip character, it must be in the encoding scheme that is used for the output.

TRUNCATE
Indicates that a graphic character string (encoded for output) is to be truncated from the right, if the data does not fit in the available space for the field in the output records. Truncation occurs at a DBCS character boundary. Without TRUNCATE, an error occurs when the output field size is too small for the data.

SMALLINT
Specifies that the output field is a 2-byte binary integer (a negative number is in two's complement notation). To use the external format, specify **INTEGER EXTERNAL**.

If the source data type is INTEGER, DECIMAL, or FLOAT (either 4-byte or 8-byte format), an error occurs when the data is greater than 32 767 or less than -32 768.

A SMALLINT output field requires 2 bytes, and the *length* option is not available.

INTEGER
Specifies that the output field is a 4-byte binary integer (a negative number is in two's complement notation).

If the original data type is DECIMAL, or FLOAT (either 4-byte or 8-byte format), an error occurs when the original data is greater than 2 147 483 647 or less than -2 147 483 648.

An INTEGER output field requires 4 bytes, and the *length* option is not available.

INTEGER EXTERNAL
Specifies that the output field is to contain a character string that represents an integer number.

(length)
Indicates the size of the output data in bytes, including a space for the sign character. When the *length* is given and the character notation does not fit in the space, an error occurs. The default is 11 characters (including a space for the sign).

If the value is negative, a minus sign precedes the numeric digits. If the output field size is larger than the length of the data, the output data is left justified and blanks are padded on the right.

If the source data type is DECIMAL, or FLOAT (either 4-byte or 8-byte format), an error occurs when the original data is greater than 2 147 483 647 or less than -2 147 483 648.

DECIMAL
Specifies that the output data is a number that is represented by the indicated decimal format (either PACKED, ZONED, or EXTERNAL). If you specify the keyword DECIMAL by itself, packed-decimal format is assumed.

PACKED
Specifies that the output data is a number that is represented by the packed-decimal format. You can use **DEC** or **DEC PACKED** as an abbreviated form of the keyword.

The packed-decimal representation of a number is of the form *ddd...ds*, where *d* is a decimal digit that is represented by 4 bits, and *s* is a 4-bit sign character (hexadecimal A, C, E, or F for a positive number, and hexadecimal B or D for a negative number).

length
Specifies the number of digits (not including the sign digit) that are to be placed in the output field. The length must be between 1 and 31. If the length is odd, the size of the output data field is (*length*+1) / 2 bytes; if even, (*length* / 2)+1 byte.

If the source data type is DECIMAL and the *length* parameter is omitted, the default length is determined by the column attribute defined on the table. Otherwise, the default length is 31 digits (16 bytes).

scale
Specifies the number of digits to the right of the decimal point. (Note that, in this case, a decimal point is not included in the output field.) The number must be an integer that is greater than or equal to zero and less than or equal to the length.

The default depends on the column attribute that is defined on the table. If the source data type is DECIMAL, the defined *scale* value is the default value; otherwise, the **default** is **0**.

If you specify the output field size as less than the length of the data, an error occurs. If the specified field size is greater than the length of data, X'0' is padded on the left.

ZONED
Specifies that the output data is a number that is represented by the zoned-decimal format. You can use DEC ZONED as an abbreviated form of the keyword.

The zoned-decimal representation of a number is of the form *znznzn...z/sn*, where *n* denotes a 4 bit decimal digit (called the numeric bits); *z* is the digit's zone (left 4 bits of a byte); *s* is the right-most operand that can be a zone (*z*) or can be a sign value (hexadecimal A, C, E, or F for a positive number, and hexadecimal B or D for a negative number).

length
Specifies the number of bytes (that is the number of decimal digits) that are placed in the output field. The length must be between 1 and 31.

If the source data type is DECIMAL and the *length* parameter is omitted, the default length is determined by the column attribute that is defined on the table. Otherwise, the default length is 31 bytes.

scale
Specifies the number of digits to the right of the decimal point. (Note that, in this case, a decimal point is not included in the output field.) The number must be an integer greater than or equal to zero and less than or equal to the length.

The default depends on the column attribute that is defined on the table. If the source data type is DECIMAL, the defined *scale* value is the default value; otherwise, the **default** is **0**.

If you specify the output field size as less than the length of the data, an error occurs. If the specified field size is greater than the length of data, X'F0' is padded on the left.

EXTERNAL
Specifies that the output data is a character string that represents a number in the form of ±*dd...d.ddd...d*, where *d* is a numeric character 0-9. (The plus sign for a positive value is omitted.)

length
Specifies the overall length of the output data (the number of characters including a sign, and a decimal point if *scale* is specified).

If the source data type is DECIMAL and the *length* parameter is omitted, the default length is determined by the column attribute that is defined on the table. Otherwise, the default length is 33 (31 numeric digits, plus a sign and a decimal point). The minimum value of *length* is 3 to accommodate the sign, one digit, and the decimal point.

scale
Specifies the number of digits to the right of the decimal point. The number must be an integer that is greater than or equal to zero and less than or equal to *length* - 2 (to allow for the sign character and the decimal point).

If the source data type is DECIMAL and the *length* parameter is omitted, the default scale is determined by the column attribute that is defined on the table. Otherwise, the **default** is **0**.

An error occurs if the character representation of a value does not fit in the given or default field size (precision). If the source data type is floating point and a data item is too small for the precision that is defined by *scale*, the value of zero (not an error) is returned.

FLOAT*(length)*
Specifies that the output data is a binary floating-point number (32-bit or single-precision FLOAT if the *length* is between one and 21 inclusive; 64-bit or double-precision FLOAT if the *length* is between 22 and 53 inclusive). If the *length* parameter is omitted, the 64-bit format is assumed (output field size is 8 bytes). Note that the *length* parameter for the FLOAT type does not represent the field size in bytes.

The format of the binary floating-point output is controlled by the global FLOAT option. The default is S/390 format (Hexadecimal Floating Point or HFP). If you specify FLOAT(IEEE), all the binary floating-point output is in IEEE format (Binary Floating Point or BFP). When you specify FLOAT(IEEE) and the source data type DOUBLE is unloaded as REAL, an error occurs if the source data cannot be expressed by the IEEE (BFP) 32-bit notation.

EXTERNAL*(length)*
Specifies that the output data is a number that is represented by a character string in floating-point notation, ±*d.ddd...dddE*±*nn*, where *d* is a numeric character (0-9) for the significant digits; *nn* after the character *E*, and the sign consists of two numeric characters for the exponent.

(length)
Specifies the total field length in bytes, including the first sign character, the decimal point, the *E* character, the second sign character, and the two-digit exponent. If the number of characters in the result is less than the specified or the default length, the result is padded to the right with blanks. The length, if specified, must be greater than or equal to 8.

The default output field size is 14 if the source data type is the 32-bit FLOAT; otherwise, the default is 24.

A FLOAT EXTERNAL output field requires a space of at least seven characters in the output record to accommodate the minimal floating point notation. Otherwise, an error occurs.

DOUBLE
Specifies that the output data is in 64-bit floating point notation. If DOUBLE is used, the *length* parameter must not be specified.

REAL Specifies that the output data is in 32-bit floating point notation. If REAL is used, the *length* parameter must not be specified.

DATE EXTERNAL
Specifies that the output field is for a character string representation of a date. The output format of date depends on the DB2 installation.

(length)
Specifies the size of the data field in bytes in the output record. A DATE EXTERNAL field requires a space of at least 10 characters. If the

space is not available, an error occurs. If the specified *length* is larger than the size of the data, blanks are padded on the right.

TIME EXTERNAL
Specifies that the output field is for a character string representation of a time. The output format of time depends on the DB2 installation.

(length)
Specifies the size of the data field in bytes in the output record. A TIME EXTERNAL field requires a space of at least eight characters. If the space is not available, a conversion error occurs. If the specified *length* is larger than the size of the data, blanks are padded on the right.

TIMESTAMP EXTERNAL
Specifies that the output field is for a character string representation of a timestamp.

(length)
Specifies the size of the data field in bytes in the output record. A TIMESTAMP EXTERNAL field requires a space of at least 19 characters. If the space is not available, an error occurs. The *length* parameter, if specified, determines the output format of the TIMESTAMP. If the specified *length* is larger than the size of the data, the field is padded on the right with the default padding character.

CONSTANT
Specifies that the output records are to have an extra field containing a constant value. The field name that is associated with the CONSTANT keyword must not coincide with a table column name (the field name is for clarification purposes only). A CONSTANT field always has a fixed length that is equal to the length of the given string.

'string'
Specifies the character string that is to be inserted in the output records at the specified or default position. A string is the required operand of the CONSTANT option. If the given string is in the form *'string'*, it is assumed to be an EBCDIC SBCS string. However, the output string for a CONSTANT field is in the specified or default encoding scheme. (That is, if the encoding scheme used for output is not EBCDIC, the SBCS CCSID conversion is applied to the given string before it is placed in output records.)

X'hex-string'
Specifies the character string in hexadecimal form, X'*hex-string*', that is to be inserted in the output records at the specified or default position. If you want to specify a CONSTANT string value in an encoding scheme other than SBCS EBCDIC, use the hexadecimal form. No CCSID conversion is performed if the hexadecimal form is used.

For a CONSTANT field, no other field selection list options should be specified.

If a CONSTANT field is inserted, it will not be included in the generated LOAD statement (the LOAD statement is generated so that the CONSTANT field is skipped).

If you specify both FORMAT DELIMITED and CONSTANT, the generated LOAD statement is not usable.

ROWID
Specifies that the output data is of type ROWID. The field type ROWID can be specified if and only if the column that is to be unloaded is of type ROWID. The keyword is provided for consistency purposes.

ROWID fields have varying length and a 2-byte binary length field is prepended to the actual data field.

For the ROWID type, no data conversion nor truncation is applied. If the output field size is too small to unload ROWID data, an error occurs.

If the source is an image copy and a ROWID column is selected, and if the page set header page is missing in the specified data set, the UNLOAD utility terminates with the error message DSNU1228I. This situation can occur when the source is an image copy data set of DSNUM that is greater than one for a nonpartitioned table space that is defined on multiple data sets.

BLOB Indicates that the column is to be unloaded as a binary large object (BLOB). No data conversion is applied to the field.

When you specify the BLOB field type, a 4-byte binary length field is placed in the output record prior to the actual data field. If the source table column can be null, a NULL indicator byte is placed before the length field.

(length)
Specifies the maximum length of the actual data field in bytes. If you specify NOPAD, it indicates the maximum allowable space for the data in the output records; otherwise, the space of the specified length is reserved for the data.

The default is the maximum length that is defined on the source table column.

TRUNCATE
Indicates that a BLOB string is to be truncated from the right, if the data does not fit in the available space for the field in the output record. For BLOB data, truncation occurs at a byte boundary. Without TRUNCATE, an error occurs when the output field size is too small for the data.

CLOB Indicates that the column is to be unloaded as a character large object (CLOB).

When you specify the CLOB field type, a 4-byte binary length field is placed in the output record prior to the actual data field. If the source table column can be null, a NULL indicator byte is placed before the length field.

If you specify the EBCDIC, ASCII, UNICODE, or CCSID options, the output data is encoded in the CCSID corresponding to the specified option, depending on the subtype of the source data (SBCS or MIXED). No conversion is applied if the subtype is BIT.

(length)
Specifies the maximum length of the actual data field in bytes. If you specify NOPAD, it indicates the maximum allowable space for the data in the output records; otherwise, the space of the specified length is reserved for the data.

The default is the maximum length that is defined on the source table column.

TRUNCATE
Indicates that a CLOB string (encoded for output) is to be truncated from the right, if the data does not fit in the available space for the field in the output record. For CLOB data, truncation occurs at a character boundary. See "Specifying TRUNCATE and STRIP options for output data" on page 658 for the truncation rules that are used in the UNLOAD utility. Without TRUNCATE, an error occurs when the output field size is too small for the data.

DBCLOB
Indicates that the column is to be unloaded as a double-byte character large object (DBCLOB).

If you specify the DBCLOB field type, a 4-byte binary length field is placed in the output record prior to the actual data field. If the source table column can be null, a NULL indicator byte is placed before the length field.

If you specify the EBCDIC, ASCII, UNICODE, or CCSID options, the output data is encoded in the CCSID corresponding to the specified option; DBCS CCSID is used.

(length)
Specifies the maximum length of the actual data field in the number of DBCS characters. If you specify NOPAD, it indicates the maximum allowable space for the data in the output records; otherwise, the space of the specified length is reserved for the data.

The default is the maximum length that is defined on the source table column.

TRUNCATE
Indicates that a DBCS string (encoded for output) is to be truncated from the right, if the data does not fit in the available space for the field in the output record. For a DBCLOB data, truncation occurs at a character (DBCS) boundary. See "Specifying TRUNCATE and STRIP options for output data" on page 658 for the truncation rules that are used in the UNLOAD utility. Without TRUNCATE, an error occurs when the output field size is too small for the data.

WHEN
Indicates which records in the table space are to be unloaded. If no WHEN clause is specified for a table in the table space, all of the records are unloaded.

The option following WHEN describes the conditions for unloading records from a table.

Data in the table can be in EBCDIC, ASCII, or Unicode. If the target table
is in Unicode and the character constants are specified in the utility control
statement as EBCDIC, the UNLOAD utility converts these constants to
Unicode. To use a constant when the target table is ASCII, specify the
hexadecimal form of the constant (instead of the character string form) in
the condition for the WHEN clause.

selection condition
Specifies a condition that is true, false, or unknown about a given row.

When the condition is true, the row qualifies for UNLOAD. When the condition is false or unknown, the row does not qualify.

The result of a selection condition is derived by application of the specified *logical operators* (AND and OR) to the result of each specified predicate. If logical operators are not specified, the result of the selection condition is the result of the specified predicate.

Selection conditions within parentheses are evaluated first. If the order of evaluation is not specified by parentheses, AND is applied before OR.

If the control statement is in the same encoding scheme as the input data, you can code character constants in the control statement. Otherwise, if the control statement is not in the same encoding scheme as the input data, you must code the condition with hexadecimal constants. For example, if the table space is in EBCDIC and the control statement is in UTF-8, use (1:1) = X'31' in the condition rather than (1:1) = '1'.

Restriction: UNLOAD cannot filter rows that contain encrypted data.

predicate

Specifies a condition that is true, false, or unknown about a row.

basic predicate

Specifies the comparison of a column with a constant. If the value of the column is null, the result of the predicate is unknown. Otherwise, the result of the predicate is true or false.

column = constant	The column is equal to the constant or labeled duration expression.
column < > constant	The column is not equal to the constant or labeled duration expression.
column > constant	The column is greater than the constant or labeled duration expression.
column < constant	The column is less than the constant or labeled duration expression.
column > = constant	The column is greater than or equal to the constant or labeled duration expression.
column < = constant	The column is less than or equal to the constant or labeled duration expression.

Note: The following alternative comparison operators are available:
!= or **¬=** for not equal.
!> or **¬>** for not greater than.
!< or **¬<** for not less than.

The symbol **¬** representing "not" is supported for compatibility purposes. Use **!** where possible.

BETWEEN predicate

Indicates whether a given value lies between two other given values that are specified in ascending order. The values can be constants or labeled duration expressions. Each of the predicate's two forms (BETWEEN and NOT BETWEEN) has an equivalent search condition, as shown in Table 127 on page 643. When relevant, the table also shows any equivalent predicates.

Table 127. BETWEEN predicates and their equivalent search conditions

Predicate	Equivalent predicate	Equivalent search condition
column BETWEEN *value1* AND *value2*	None	(*column* >= *value1* AND *column* <= *value2*)
column NOT BETWEEN *value1* AND *value2*	NOT(*column* BETWEEN *value1* AND *value2*)	(*column* < *value1* OR *column* > *value2*)

Note: The values can be constants or labeled duration expressions.

For example, the following predicate is true for any row when salary is greater than or equal 10000 and less than or equal to 20000:

```
SALARY BETWEEN 10000 AND 20000
```

IN predicate
Specifies that a value is to be compared with a set of values. In the IN predicate, the second operand is a set of one or more values that are specified by constants. Each of the predicate's two forms (IN and NOT IN) has an equivalent search condition, as shown in Table 128.

Table 128. IN predicates and their equivalent search conditions

Predicate	Equivalent search condition
value1 IN (*value1*, *value2*,..., *valuen*)	(*value1* = *value2* OR ... OR *value1* = *valuen*)
value1 NOT IN (*value1*, *value2*,..., *valuen*)	*value1* ¬= *value2* AND ... AND *value1* ¬= *valuen*)

Note: The values can be constants or labeled duration expressions.

For example, the following predicate is true for any row whose employee is in department D11, B01, or C01:

```
WORKDEPT IN ('D11', 'B01', 'C01')
```

LIKE predicate
Specifies the qualification of strings that have a certain pattern.

Within the pattern, a percent sign or underscore can have a special meaning, or it can represent the literal occurrence of a percent sign or underscore. To have its literal meaning, it must be preceded by an *escape character*. If it is not preceded by an escape character, it has its special meaning. The underscore character (_) represents a single, arbitrary character. The percent sign (%) represents a string of zero or more arbitrary characters.

The ESCAPE clause designates a single character. That character, and only that character, can be used multiple times within the pattern as an escape character. When the ESCAPE clause is omitted, no character serves as an escape character, so that percent signs and underscores in the pattern always have their special meanings.

The following rules apply to the use of the ESCAPE clause:

- The ESCAPE clause cannot be used if *x* is mixed data.
- If *x* is a character string, the data type of the string constant must be character string. If *x* is a graphic string, the data type of the string constant must be graphic string. In both cases, the length of the string constant must be 1.

- The pattern must not contain the escape character except when followed by the escape character, '%' or '_'. For example, if '+' is the escape character, any occurrence of '+' other than '++', '+_', or '+%' in the pattern is an error.

When the pattern does not include escape characters, a simple description of its meaning is:

- The underscore sign (_) represents a single arbitrary character.
- The percent sign (%) represents a string of zero or more arbitrary characters.
- Any other character represents a single occurrence of itself.

Let *x* denote the column that is to be tested and *y* the pattern in the string constant. The following rules apply to predicates of the form ″x LIKE y...″. If NOT is specified, the result is reversed.

- When *x* and *y* are both neither empty nor null, the result of the predicate is true if *x* matches the pattern in *y* and false if *x* does not match the pattern in *y*.
- When *x* or *y* is null, the result of the predicate is unknown.
- When *y* is empty and *x* is not empty, the result of the predicate is false.
- When *x* is empty and *y* is not empty, the result of the predicate is false unless *y* consists only of one or more percent signs.
- When *x* and *y* are both empty, the result of the predicate is true.

The pattern string and the string that is to be tested must be of the same type. That is, both *x* and *y* must be character strings, or both *x* and *y* must be graphic strings. When *x* and *y* are graphic strings, a character is a DBCS character. When *x* and *y* are character strings and *x* is not mixed data, a character is an SBCS character and *y* is interpreted as SBCS data regardless of its subtype. The rules for mixed-data patterns are described under “Strings and patterns” on page 645.

Strings and patterns

The string *y* is interpreted as a sequence of the minimum number of substring specifiers such that each character of *y* is part of exactly one substring specifier. A substring specifier is an underscore, a percent sign, or any non-empty sequence of characters other than an underscore or percent sign.

The string *x* matches the pattern *y* if a partitioning of *x* into substrings exists, such that:

- A substring of *x* is a sequence of zero or more contiguous characters, and each character of *x* is part of exactly one substring.
- If the *n*th substring specifier is an underscore, the *n*th substring of *x* is any single character.
- If the *n*th substring specifier is a percent sign, the *n*th substring of *x* is any sequence of zero or more characters.
- If the *n*th substring specifier is neither an underscore nor a percent sign, the *n*th substring of *x* is equal to that substring specifier and has the same length as that substring specifier.
- The number of substrings of *x* is the same as the number of substring specifiers.

When escape characters are present in the pattern string, an underscore, percent sign, or escape character represents a single occurrence of itself if and only if it is preceded by an odd number of successive escape characters.

Mixed data patterns: If *x* is mixed data, the pattern is assumed to be mixed data, and its special characters are interpreted as follows:

- A single-byte underscore refers to one single-byte character; a double-byte underscore refers to one double-byte character.
- A percent sign, either single-byte or double-byte, refers to any number of characters of any type, either single-byte or double-byte.
- Redundant shift bytes in *x* or *y* are ignored.

NULL predicate
Specifies a test for null values.

If the value of the column is null, the result is true. If the value is not null, the result is false. If NOT is specified, the result is reversed. (That is, if the value is null, the result is false, and if the value is not null, the result is true.)

labeled duration expression
Specifies an expression that begins with special register CURRENT DATE or special register CURRENT TIMESTAMP (the forms CURRENT_DATE and CURRENT_TIMESTAMP are also acceptable). This special register can be followed by arithmetic operations of addition or subtraction. These operations are expressed by using numbers that are followed by one of the seven duration keywords: YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS, or

MICROSECONDS. (The singular form of these keywords is also acceptable: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, and MICROSECOND.)

Utilities always evaluate a *labeled duration expression* as a timestamp and implicitly convert to a date if the comparison is with a date column.

Incrementing and decrementing CURRENT DATE: The result of adding a duration to a date, or of subtracting a duration from a date, is itself a date. (For the purposes of this operation, a month denotes the equivalent of a calendar page. Adding months to a date, then, is like turning the pages of a calendar, starting with the page on which the date appears.) The result must fall between the dates January 1, 0001 and December 31, 9999 inclusive. If a duration of years is added or subtracted, only the year portion of the date is affected. The month is unchanged, as is the day, unless the result would be February 29 of a non-leap-year. In this situation, the day portion of the result is set to 28.

Similarly, if a duration of months is added or subtracted, only months and, if necessary, years are affected. The day portion of the date is unchanged unless the result would be invalid (September 31, for example). In this case the day is set to the last day of the month.

Adding or subtracting a duration of days affects the day portion of the date, and potentially the month and year.

Date durations, whether positive or negative, can also be added to and subtracted from dates. As with labeled durations, the result is a valid date.

When a positive date duration is added to a date, or a negative date duration is subtracted from a date, the date is incremented by the specified number of years, months, and days.

When a positive date duration is subtracted from a date, or a negative date duration is added to a date, the date is decremented by the specified number of days, months, and years.

Adding a month to a date gives the same day one month later, unless that day does not exist in the later month. In that case, the day in the result is set to the last day of the later month. For example, January 28 plus one month gives February 28; one month added to January 29, 30, or 31 results in either February 28 or, for a leap year, February 29. If one or more months is added to a given date and then the same number of months is subtracted from the result, the final date is not necessarily the same as the original date.

The order in which labeled date durations are added to and subtracted from dates can affect the results. When you add labeled date durations to a date, specify them in the order of YEARS + MONTHS + DAYS. When you subtract labeled date durations from a date, specify them in the order of DAYS - MONTHS - YEARS. For example, to add one year and one day to a date, specify the following code:

```
CURRENT DATE + 1 YEAR + 1 DAY
```

To subtract one year, one month, and one day from a date, specify the following code:

```
CURRENT DATE - 1 DAY - 1 MONTH - 1 YEAR
```

Incrementing and decrementing timestamps: The result of adding a duration to a timestamp, or of subtracting a duration from a timestamp, is itself a timestamp. Date and time arithmetic is performed as previously defined, except that an overflow or underflow of hours is carried into the date part of the result, which must be within the range of valid dates.

Instructions for running UNLOAD

To run UNLOAD, you must:

1. Read "Before running UNLOAD" in this section.
2. Prepare the necessary data sets, as described in "Data sets that UNLOAD uses."
3. Create JCL statements, by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15. (For examples of JCL for UNLOAD, see "Sample UNLOAD control statements" on page 662.)
4. Prepare a utility control statement, that specifies the options for the tasks that you want to perform, as described in "Instructions for specific tasks" on page 648.
5. Check the compatibility table in "Concurrency and compatibility for UNLOAD" on page 660 if you want to run other jobs concurrently on the same target objects.
6. Plan for restart if the UNLOAD job doesn't complete, as described in "Terminating or restarting UNLOAD" on page 660.
7. Run UNLOAD by using one of the methods that are described in Chapter 3, "Invoking DB2 online utilities," on page 15.

Before running UNLOAD

If you plan to run UNLOAD on encrypted data, do not use the WHEN statement to filter encrypted fields; UNLOAD cannot filter rows that contain encrypted data

Data sets that UNLOAD uses

Table 129 lists the data sets that UNLOAD uses. The table lists the DD name that is used to identify the data set, a description of the data set, and an indication of whether it is required. Include statements in your JCL for each required data set and any optional data sets that you want to use.

Table 129. Data sets that UNLOAD uses

Data set	Description	Required?
SYSIN	Input data set that contains the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
SYSPUNCH	One or more work data sets that contain the generated LOAD statements for subsequently reloading the data. The default DD name is PUNCHDDN.	No[1]
Unload data set	One or more work data sets that contain the unloaded table rows. The default DD name is SYSREC.	Yes

Table 129. Data sets that UNLOAD uses (continued)

Data set	Description	Required?

Notes:
1. Required if you request that UNLOAD generate LOAD statements by specifying PUNCHDDN in the utility control statement.

The following object is named in the utility control statement and does not require a DD statement in the JCL:

Table space
Table space that is to be unloaded. (If you want to unload only one partition of a table space, you must specify the PART option in the control statement.)

Instructions for specific tasks

The following tasks are described here:

Unloading partitions

If the source table space is partitioned, use one of the following mutually exclusive methods to select the partitions to unload:

- Use the LIST keyword with a LISTDEF that contains PARTLEVEL specifications. Partitions can be either included or excluded by the use of the INCLUDE and the EXCLUDE features of LISTDEF.
- Specify the PART keyword to select a single partition or a range of partitions.

With either method, the unloaded data can be stored in a single data set for all selected partitions or in one data set for each selected partition. If you want to unload to a single output data set, specify a DD name to UNLDDN. If you want to unload into multiple output data sets, specify a template name that is associated with the partitions. You can process multiple partitions in parallel if the TEMPLATE definition contains the partition as a variable, for example &PA.

You cannot specify multiple output data sets with the FROMCOPY or the FROMCOPYDDN option.

Selecting tables and rows to unload

If a table space contains multiple tables, you can select specific tables to unload by using the FROM TABLE specification clauses. If you specify one or more FROM TABLE clauses for a table space, only the qualified rows from the specified tables

are unloaded. You can specify a maximum of one FROM TABLE clause per table. If you do not specify at least one FROM TABLE clause, the rows from all the tables in the table space are unloaded.

Within a FROM TABLE clause, you can specify one or more of the following criteria:
- Row and column selection criteria by using the field specification list
- Row selection conditions by using the WHEN specification clause
- Row sampling specifications

Important: When an incremental image copy is taken of a table space, rows might be updated or moved if the SHRLEVEL CHANGE option is specified. As a result, data that is unloaded from such a copy might contain duplicates of these rows.

Selecting and ordering columns to unload

Use a field specification list in a FROM TABLE clause to unload specified columns in the listed order. If you omit a field specification list, all the columns in the row are unloaded in the order of the columns that are defined on the table.

You can specify a format conversion option for each field in the field specification list.

If you select a LOB column in a list of field specifications or select a LOB column by default (by omitting a list of field specifications), LOB data is materialized in the output. However, you cannot select LOB columns from image copy data sets.

Unloading data from image copy data sets

In addition to unloading data from table spaces and partitions, you can also unload data from one or more image copy data sets. If you use the SYSTEMPAGES YES option on the COPY utility, you can use UNLOAD to process rows of compressed data or image copies from different versions. See SYSTEMPAGES in the "Option descriptions" on page 107 for the COPY utility.

Unload rows from a single image copy data set by specifying the FROMCOPY option in the UNLOAD control statement. Specify the FROMCOPYDDN option to unload data from one or more image copy data sets that are associated with the specified DD name. Use an image copy that contains the page set header page when you are unloading a ROWID column; otherwise the unload fails.

The source image copy data set must have been created by one of the following utilities:
- COPY
- COPYTOCOPY
- LOAD inline image copy
- MERGECOPY
- REORG TABLESPACE inline image copy
- DSN1COPY

UNLOAD accepts full image copies, incremental image copies, and a copy of pieces as valid input sources.

The UNLOAD utility supports image copy data sets for a single table space. The table space name must be specified in the TABLESPACE option. The specified table space must exist when you run the UNLOAD utility. (That is, the table space cannot have been dropped since the image copy was taken.)

Use the FROMCOPYDDN option to concatenate the copy of table space partitions under a DD name to form a single input data set image. When you use the FROMCOPYDDN option, concatenate the data sets in the order of the data set number; the first data set must be concatenated first. If the data sets are concatenated in the wrong order or if different generations of image copies are concatenated, the results might be unpredictable. For example, if the most recent image copy data sets and older image copies are intermixed, the results might be unpredictable.

You can use the FROMCOPYDDN option to concatenate a full image copy and incremental image copies for a table space, a partition, or a piece, but duplicate rows are also unloaded in this situation. Instead, consider using MERGECOPY to generate an updated full image copy as the input to the UNLOAD utility.

You can select specific rows and columns to unload just as you would for a table space. However, you can unload only rows that contain LOB columns when the LOB columns are not included in a field specification list. If you use an image copy that does not contain the page set header page when unloading a ROWID column, the unload fails.

If you use the FROMCOPY or the FROMCOPYDDN option, you can specify only one output data set.

If an image copy is created by an inline copy operation (LOAD or REORG TABLESPACE), the image copy can contain duplicate pages. If duplicate pages exist, the UNLOAD utility issues a warning message, and all the qualified rows in duplicate pages are unloaded into the output data set.

If you specify a dropped table on the FROM TABLE option, the UNLOAD utility terminates with return code 4. If you do not specify a FROM TABLE option and if an image copy contains rows from dropped tables, UNLOAD ignores these rows. When you specify either a full or incremental copy of partitions of a segmented table space that consists of multiple data sets in the FROMCOPY option, be careful when applying a mass delete to a table in the table space before you create the copy. If a mass delete of a table occurs, the utility unloads deleted rows if the space map pages that indicate the mass delete are not included in the data set that corresponds to the specified copy. Where possible, use the FROMCOPYDDN option to concatenate the copy of table space partitions.

If an image copy contains a table to which ALTER ADD COLUMN was applied after the image copy was taken, the UNLOAD utility sets the system or user-specified default value for the added column when the data is unloaded from such an image copy.

Converting data with the UNLOAD utility

You can convert one data type to another compatible data type by using the UNLOAD utility.[5] For example, you can convert columns of a numeric type (SMALLINT, INTEGER, FLOAT, DOUBLE, REAL, and DECIMAL) from the DB2 internal format to the S/390 or an external format.

When you unload a floating-point type column, you can specify the binary form of the output to either the S/390 format (hexadecimal floating point, or HFP), or the IEEE format (binary floating point, or BFP).

5. The source type is used for user-defined distinct types.

You can also convert a varying-length column to a fixed-length output field, with or without padding characters. In either case, unless you explicitly specify a fixed-length data type for the field, the data itself is treated as a varying-length data, and a length field is appended to the data.

For certain data types, you can unload data into fields with a smaller length by using the TRUNCATE or STRIP options. In this situation, if a character code conversion is applied, the length of the data in bytes might change due to the code conversion. The truncation operation is applied after the code conversion.

You can perform character code conversion on a character type field, including converting numeric columns to the external format and the CLOB type. Be aware that when you apply a character code conversion for mixed-data fields, the length of the result string in bytes can be shorter or longer than the length of the source string. Character type data is always converted if you specify any of the character code conversion options (EBCDIC, ASCII, UNICODE, or CCSID).

DATE, TIME, or TIMESTAMP column types are always converted into the external formats based on the DATE, TIME, and TIMESTAMP formats of your installation.

Specifying output field types

An output field can have a different data type from the one that is defined on a source table column if the data types are compatible. The UNLOAD utility follows the general DB2 rules and conventions on the data type attributes and the compatibility among the data types, as described in Chapter 2 of *DB2 SQL Reference*.

If you specify a data type in the UNLOAD control statement, the field type information is included in the generated LOAD utility statement. For specific data type compatibility information, refer to Table 130, Table 131 on page 652, and Table 132 on page 652. These tables show the compatibility of the data type of the source column (input data type) with the data type of the output field (output data type). A Y indicates that the input data type can be converted to the output data type.

Table 130 shows the compatibility of converting numeric data types.

Table 130. Compatibility of converting numeric data types

	Output data types				
Input data types	**SMALLINT**	**INTEGER (external)**	**DECIMAL (external)**	**FLOAT (external)**	**DOUBLE or REAL**
SMALLINT	Y	Y[1]	Y[1]	Y[1]	Y
INTEGER	Y[2]	Y[1]	Y[1]	Y[1]	Y
DECIMAL	Y[2]	Y[1, 2]	Y[1]	Y[1]	Y
FLOAT, DOUBLE, or REAL	Y[2]	Y[1, 2]	Y[1, 2]	Y[1]	Y

Notes:

1. Subject to the CCSID conversion, if specified (EXTERNAL case). For more information about CCSID, see "CCSID" on page 619.
2. Potential overflow (conversion error).

Table 131 on page 652 shows the compatibility of converting character data types.

Table 131. Compatibility of converting character data types

Input data types	Output data types							
	BLOB	**CHAR**	**VAR-CHAR**	**CLOB**	**GRAPHIC**	**GRAPHIC EXTERNAL**	**VAR-GRAPHIC**	**DBCLOB**
BLOB	Y	N	N	N	N	N	N	N
CLOB	N	Y [1, 2]	Y [1, 2]	Y	N	N	N	N
DBCLOB	N	N	N	N	Y [1, 2]	Y [1, 2, 3]	Y [1, 2]	Y [1]
CHAR	N	Y [1]	Y [1]	Y [1, 4]	N	N	N	N
VARCHAR or LONG VARCHAR	N	Y[1,2]	Y[1]	Y[1, 4]	N	N	N	N
GRAPHIC	N	N	N	N	Y [1]	Y [1, 3]	Y [1]	Y [1]
VAR-GRAPHIC or LONG VAR-GRAPHIC	N	N	N	N	Y [1, 2]	Y [1, 2, 3]	Y [1]	Y [1]

Notes:

1. Subject to the CCSID conversion, if specified.
2. Results in an error if the field length is too small for the data unless you specify the TRUNCATE option. Note that a LOB has a 4-byte length field; any other varying-length type has a 2-byte length field.
3. Only in the EBCDIC output mode.
4. Not applicable to BIT subtype data.

Table 132 shows the compatibility of converting time data types.

Table 132. Compatibility of converting time data types

Input data types	Output data types		
	DATE EXTERNAL	**TIME EXTERNAL**	**TIMESTAMP EXTERNAL**
DATE	Y[1]	N	Y[1, 2]
TIME	N	Y[1]	N
TIMESTAMP	Y[1, 3]	Y[1, 3]	Y[1]

Notes:

1. Subject to the CCSID conversion, if specified.
2. Zeros in the time portion.
3. DATE or TIME portion of the timestamp.

Specifying output field positioning and size

By default, output data is always placed in an output record in the order of the defined columns over the selected tables. You can choose to specify the order of the output fields by using a list of field specifications.

Use the POSITION option to specify field position in the output records. You can also specify the size of the output data field by using the *length* parameter for a particular data type. The *length* parameter must indicate the size of the actual data field. The *start* parameter of the POSITION option indicates the starting position of a field, including the NULL indicator byte (if the field can be null) and the length field (if the field is varying length).

Using the POSITION parameter, the length parameter, or both can restrict the size of the data field in the output records. Use care when specifying the POSITION and length parameters, especially for nullable fields and varying length fields. If a conflict exists between the length parameter and the size of the field in the output record that is specified by the POSITION parameters, DB2 issues an error message, and the UNLOAD utility terminates. If an error occurs, the count of the number of records in error is incremented. See the description of the MAXERR option under "MAXERR" on page 622 for more information.

If you specify a length parameter for a varying-length field and you also specify the NOPAD option, *length* indicates the maximum length of data that is to be unloaded. Without the NOPAD option, UNLOAD reserves a space of the given *length* instead of the maximum data size.

If you explicitly specify start parameters for certain fields, they must be listed in ascending order in the field selection list. Unless you specify HEADER NONE for the table, a fixed-length record header is placed at the beginning of each record for the table, and the start parameter must not overlap the record header area.

The TRUNCATE option is available for certain output field types. See "FROM-TABLE-spec " on page 623 and "Specifying TRUNCATE and STRIP options for output data" on page 658 for more information. For the output field types where the TRUNCATE option is not applicable, enough space must be provided in the output record for each field. The output field layouts are summarized in "Determining the layout of output fields."

For information about errors that can occur at the record level due to the field specifications, see "Interpreting field specification errors" on page 660.

Determining the layout of output fields

To determine the layout of a fixed-length field that cannot be null, see the layout diagram in Figure 107. This diagram shows that the data field begins at a specified position, or at the next byte position past the end of the previous data field. The data field then continues for the specified length or the length of the column in the table definition. For GRAPHIC EXTERNAL data, shift-in and shift-out characters are inserted before and after the data.

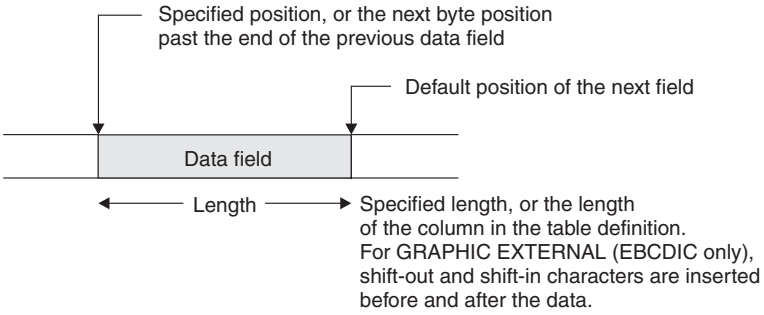


Figure 107. Layout of a fixed-length field (NOT NULL)

To determine the layout of a fixed-length field that can be null, see the layout diagram in Figure 108 on page 654. This diagram shows that a null indicator byte is stored before the data field, which begins at the specified position or at the next byte position past the end of the previous data field.

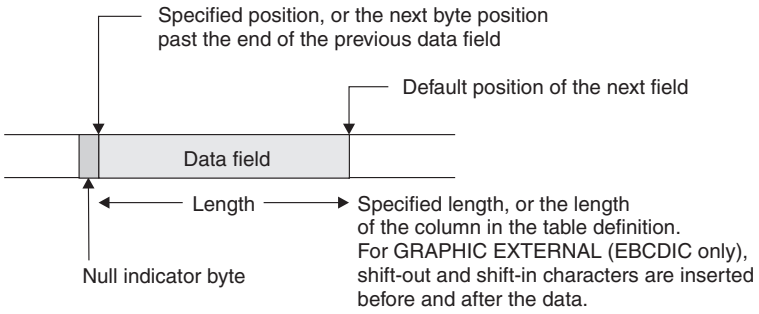


Figure 108. Layout of a nullable fixed-length field

If you are running UNLOAD with the NOPAD option and need to determine the layout of a varying-length field that cannot be null, see the layout diagram in Figure 109. This diagram shows that a length field, which contains the actual length of the data, is stored before the data field. The length field begins at the specified position or at the next byte position past the end of the previous data field.

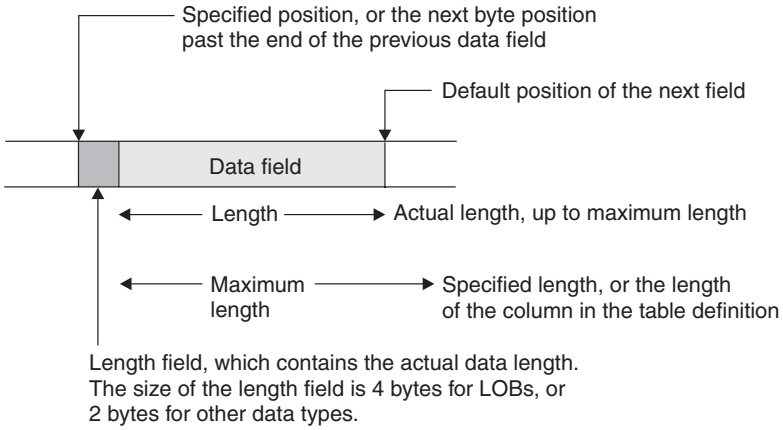


Figure 109. Layout of a varying-length field (NOT NULL) with the NOPAD option

If you are running UNLOAD without the NOPAD option and need to determine the layout of a varying-length field that cannot be null, see the layout diagram in Figure 110 on page 655. This diagram shows that the length field is stored before the data field and that the padding is after the data field.

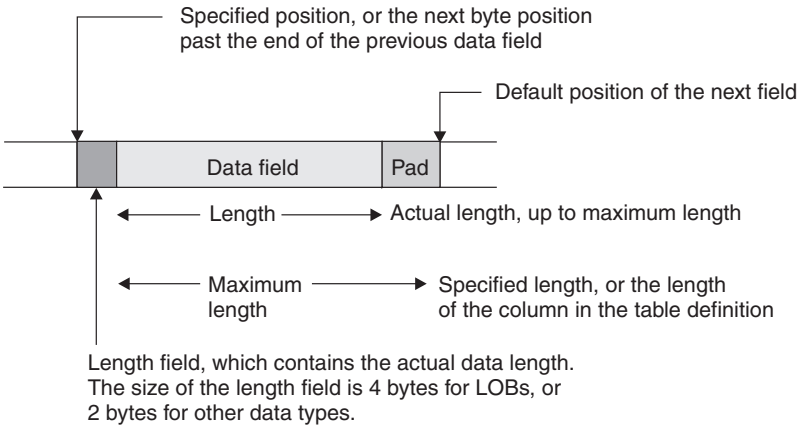


Figure 110. Layout of a varying-length field (NOT NULL) without the NOPAD option

If you are running UNLOAD with the NOPAD option and need to determine the layout of a varying-length field that can be null, see the layout diagram in Figure 111. This diagram shows that the null indicator is stored before the length field, which is stored before the data field. The length field begins at the specified position or at the next byte position past the end of the previous data field.

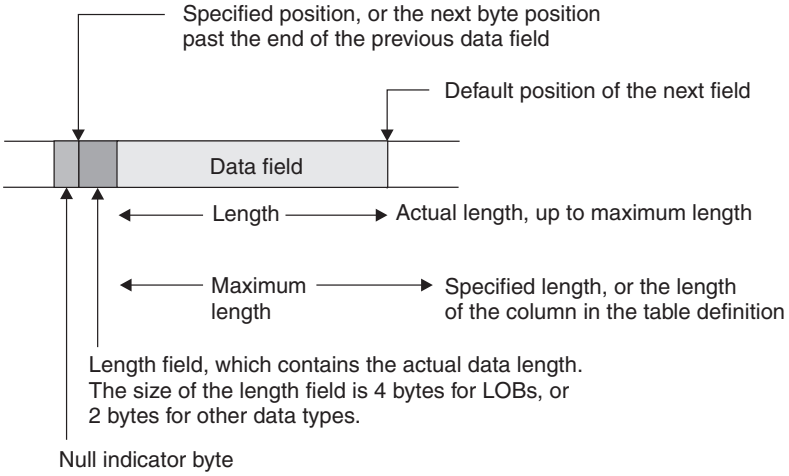


Figure 111. Layout of a nullable varying-length field with the NOPAD option

If you are running UNLOAD without the NOPAD option and need to determine the layout of a varying-length field that can be null, see the layout diagram in Figure 112 on page 656. This diagram shows that the null indicator is stored before the length field, which is stored before the data field, which has padding at the end.

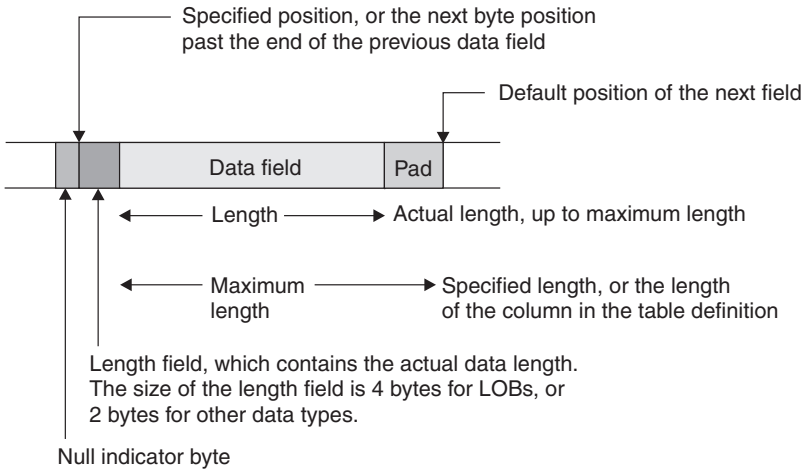


Figure 112. Layout of a nullable varying-length field without the NOPAD option

Unloading delimited files

You can use the DELIMITED option to specify that UNLOAD is to produce an output file in delimited format. All fields in the output data set are either in character string or numeric external format. Each column is separated from the next by a column delimiter, and character strings are marked by character string delimiters. For more information on delimited files see Appendix F, "Delimited file format," on page 899.

Recommendation: If a delimited file is to be transferred to or from a platform other than z/OS or between DB2 for z/OS systems that use different EBCDIC or ASCII CCSIDs, use Unicode as the encoding scheme for the delimited file. Using Unicode avoids possible CCSID translation problems.

You are responsible for ensuring that the chosen delimiters are not part of the data in the file. If the delimiters are part of the file's data, unexpected errors can occur.

Restrictions: The following general restrictions apply to the use of delimiters:

- You cannot specify the same character for more than one type of delimiter (COLDEL, CHARDEL, and DECPT).
- You can specify a character constant for a delimiter if the utility control statement is coded in the same encoding scheme as the output file. For example, the utility control statement is coded in Unicode and the output data is also coded in Unicode.
- Use the hex representation for non-default delimiters if the utility control statement is coded in a different encoding scheme than the output file. For example, the utility control statement is coded in Unicode and the output file is coded in EBCDIC. In this case, if you do not use the hex representation for the non-default delimiters, the results can be unpredictable.
- You cannot specify HEADER OBID and ROWID for output fields in delimited output format. Because a header is not allowed, output must be from a single table.
- When you specify the DELIMITED option, the utility ignores the POSITION keyword. The utility overrides field data type specifications according to the specifications of the delimited format. (For example, length values for CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, CLOB, DBCLOB, and BLOB data are the delimited lengths of each field in the output data set, and the utility unloads all numeric types in external format.)

- You cannot specify a binary 0 (zero) for any delimiter.
- No null byte is present for a delimited output file. A null value is indicated by the absence of a cell value where one would normally occur. For example, two successive column delimiters or a missing column at the end of a record indicate a null value.
- You cannot use the default decimal point as a character string delimiter (CHARDEL) or a column string delimiter (COLDEL).
- Shift-in and shift-out characters cannot be specified as EBCDIC MBCS delimiters.
- In the DBCS environment, the pipe character (|) is not supported.
- If the output is coded in ASCII or Unicode, you cannot specify any of the following values for any delimiter: X'0A', X'0D', X'2E'.
- If the output is coded in EBCDIC, you cannot specify any of the following values for any delimiter: X'15', X'0D', X'25'.
- If the output is coded in EBCDIC DBCS or MBCS, you cannot specify any of the following values for character string delimiters: X'0D', X'15', X'25', X'4B'.

Table 32 on page 245 lists by encoding scheme the default hex values for the delimiter characters.

Table 133. Default delimiter values for different encoding schemes

Character	EBCDIC SBCS	EBCDIC DBCS/MBCS	ASCII/Unicode SBCS	ASCII/Unicode MBCS
Character string delimiter	X'7F'	X'7F'	X'22'	X'22'
Decimal point character	X'4B'	X'4B'	X'2E'	X'2E'
Column delimiter	X'6B'	X'6B'	X'2C'	X'2C'

In most EBCDIC code pages, the hex values in Table 32 on page 245 represent a double quotation mark(") for the character string delimiter, a period(.) for the decimal point character, and a comma(,) for the column delimiter.

Table 33 on page 245 lists by encoding scheme the maximum allowable hex values for any delimiter character.

Table 134. Maximum delimiter values for different encoding schemes

Encoding scheme	Maximum allowable value
EBCDIC SBCS	None
EBCDIC DBCS/MBCS	X'3F'
ASCII/Unicode SBCS	None
ASCII/Unicode MBCS	X'7F'

Table 135 on page 658 identifies the acceptable data type forms for the delimited file format that the LOAD and UNLOAD utilities use.

Table 135. Acceptable data type forms for delimited files

Data type	Acceptable form for loading a delimited file	Form that is created by unloading a delimited file
CHAR, VARCHAR	A delimited or non-delimited character string	Character data that is enclosed by character delimiters. For VARCHAR, length bytes do not precede the data in the string.
GRAPHIC (any type)	A delimited or non-delimited character stream	Data that is unloaded as a delimited character string. For VARGRAPHIC, length bytes do not precede the data in the string.
INTEGER (any type)	A stream of characters that represents a number in EXTERNAL format	Numeric data in external format.
Decimal (any type)	A character stream that represents a number in EXTERNAL format	A string of characters that represents a number.
FLOAT	Representation of a number in the range `-7.2E + 75` to `7.2E + 75`in EXTERNAL format	A string of characters that represents a number in floating-point notation.
BLOB, CLOB	A delimited or non-delimited character string	Character data that is enclosed by character delimiters. Length bytes do not precede the data in the string.
DBCLOB	A delimited or non-delimited character string	Character data that is enclosed by character delimiters. Length bytes do not precede the data in the string.
DATE	A delimited or non-delimited character string that contains a date value in EXTERNAL format	A string of characters that represents a date.
TIME	A delimited or non-delimited character string that contains a time value in EXTERNAL format	A string of characters that represents a time.
TIMESTAMP	A delimited or non-delimited character string that contains a timestamp value in EXTERNAL format	A string of characters that represents a timestamp.

Specifying TRUNCATE and STRIP options for output data

You can unload certain types of data into output fields that are shorter than the length of the output data. This data truncation occurs only when you explicitly specify the TRUNCATE option. Any CCSID conversion is applied first, and then truncation is applied to encoded data for output.

For bit strings, truncation occurs at a byte boundary. For character type data, truncation occurs at a character boundary (a multi-byte character is not split). If a mixed-character type data is truncated in an output field of fixed size, the truncated string can be shorter than the specified field size. In this case, blanks in the output CCSID are padded to the right. If the output data is in EBCDIC for a mixed-character type field, truncation preserves the SO (shift-out) and the SI (shift-in) characters around a DBCS substring.

The TRUNCATE option of the UNLOAD utility truncates string data, and it has a different purpose than the SQL TRUNCATE scalar function.

For VARCHAR and VARGRAPHIC output fields, in addition to the TRUNCATE option, the STRIP option is provided to remove the specified characters, or the leading blanks, the trailing blanks, or both. The strip operation is applied on the encoded data for output. If both the TRUNCATE and STRIP options are specified, the truncation operation is applied first, and then strip is applied. For example, the output for an UNLOAD job in which you specify both the TRUNCATE and STRIP options for a VARCHAR(5) output field is shown in Table 136. In this table, an underscore represents a character that is to be stripped. In all cases, the source string is first truncated to `'_ABC_'` (a five-character string to fit in the VARCHAR(5) field), and then the strip operation is applied.

Table 136. Results of specifying both the TRUNCATE and STRIP options for UNLOAD

Specified STRIP option	Source string	Truncated string	Output string	Specified *length*
STRIP BOTH	`'_ABC_DEF'`	`'_ABC_'`	`'ABC'`	3
STRIP LEADING	`'_ABC_DEF'`	`'_ABC_'`	`'ABC_'`	4
STRIP TRAILING	`'_ABC_DEF'`	`'_ABC_'`	`'_ABC'`	4

Generating LOAD statements

To enable reloading the unloaded data into either the original table or different tables, a LOAD utility statement is generated and written to the SYSPUNCH DD name or to the DD name that is specified by PUNCHDDN.

The generated LOAD statement includes WHEN and INTO TABLE specifications that identify the table where the rows are to be reloaded, unless the HEADER NONE option was specified in the UNLOAD control statement. You need to edit the generated LOAD statement if you intend to load the UNLOAD output data into different tables than the original ones.

If multiple table spaces are to be unloaded and you want UNLOAD to generate LOAD statements, you must specify a physically distinct data set for each table space to PUNCHDDN by using a template that contains the table space as a variable (&TS.).

If PUNCHDDN is not specified and the SYSPUNCH DD name does not exist, the LOAD statement is not generated.

Unloading compressed data

You can unload compressed rows from an image copy data set only when the dictionary for decompression has been retrieved. If a row is compressed and the dictionary the dictionary pages have not been read when the row is encountered, the UNLOAD utility ignores this row, issues a warning message, and increments the error count. If the error count reaches the limit that is specified by the MAXERR option, UNLOAD terminates with an error message.

If the image copy data set is an incremental copy or a copy of pieces that does not contain a dictionary, the FROMCOPYDDN option can be used for a DD name to concatenate the data set with the corresponding full image copy that contains the dictionary. If SYSTEMPAGES YES is used, a dictionary will always be available in the incremental copies or pieces. For more information, see “FROMCOPYDDN” on page 617.

Interpreting field specification errors

If the UNLOAD utility detects any inconsistency relating to the field specification, including a problem in data conversion or encoding while unloading a row, DB2 issues an error message. If the MAXERR option specifies a number that is greater than zero, the UNLOAD utility continues processing until the total number of the records in error reaches the specified MAXERR number. DB2 issues one message for each record in error and does not unload the record. For information about specific error messages, see *DB2 Messages*.

Terminating or restarting UNLOAD

If you terminate UNLOAD by using the TERM UTILITY command during the unload phase, the output records are not erased. The output data set remains incomplete until you either delete it or restart the utility job.

For instructions on restarting a utility job, see "Restarting an online utility" on page 43. When the source is one or more table spaces, you can restart the UNLOAD job at the partition level or at the table space level when data is unloaded from multiple table spaces by using the LIST option. When you restart a terminated UNLOAD job, processing begins with the table spaces or partitions that had not yet been completed. For a table space or partitions that were being processed at termination, UNLOAD resets the output data sets and processes those table space or partitions again.

When the source is one or more image copy data sets (when FROMCOPY or FROMCOPYDDN is specified), UNLOAD always starts processing from the beginning.

Concurrency and compatibility for UNLOAD

DB2 treats Individual data partitions as distinct source objects. Utilities that operate on different partitions of the same table space are compatible.

Claims and drains: Table 137 shows which claim classes UNLOAD drains and the restrictive states that the utility sets.

Table 137. Claim classes of UNLOAD operations

Target	UNLOAD	UNLOAD PART
Table space or physical partition of a table space with SHRLEVEL REFERENCE	DW/UTRO	DW/UTRO
Table space or physical partition of a table space with SHRLEVEL CHANGE	CR/UTRW	CR/UTRW
Image copy*	CR/UTRW	CR/UTRW

Legend:
- DW: Drain the write claim class, concurrent access for SQL readers
- UTRO: Utility restrictive state, read-only access allowed
- CR: Claim read, concurrent access for SQL writers and readers
- UTRW: Utility restrictive state; read-write access allowed

Note: * If the target object is an image copy, the UNLOAD utility applies CR/UTRW to the corresponding table space or physical partitions to prevent the table space from being dropped while data is being unloaded from the image copy, even though the UNLOAD utility does not access the data in the table space.

Compatibility: The compatibility of the UNLOAD utility and the other utilities on the same target objects are shown in Table 138. If the SHRLEVEL REFERENCE option is specified, only SQL read operations are allowed on the same target objects; otherwise SQL INSERT, DELETE, and UPDATE are also allowed. If the target object is an image copy, INSERT, DELETE, and UPDATE are always allowed on the corresponding table space. In any case, DROP or ALTER cannot be applied to the target object while the UNLOAD utility is running.

Table 138. Compatibility of UNLOAD with other utilities

Action	UNLOAD SHRLEVEL REFERENCE	UNLOAD SHRLEVEL CHANGE	FROM IMAGE COPY
CHECK DATA DELETE NO	Yes	Yes	Yes
CHECK DATA DELETE YES	No	No	Yes
CHECK INDEX	Yes	Yes	Yes
CHECK LOB	Yes	Yes	Yes
COPY INDEXSPACE	Yes	Yes	Yes
COPY TABLESPACE	Yes	Yes	Yes*
DIAGNOSE	Yes	Yes	Yes
LOAD SHRLEVEL CHANGE	No	Yes	Yes
LOAD SHRLEVEL NONE	No	No	Yes
MERGECOPY	Yes	Yes	No
MODIFY RECOVERY	Yes	Yes	No
MODIFY STATISTICS	Yes	Yes	Yes
QUIESCE	Yes	Yes	Yes
REBUILD INDEX	Yes	Yes	Yes
RECOVER (no options)	No	No	Yes
RECOVER ERROR RANGE	No	No	Yes
RECOVER TOCOPY or TORBA	No	No	Yes
REORG INDEX	Yes	Yes	Yes
REORG TABLESPACE UNLOAD CONTINUE or PAUSE	No	No	Yes
REORG TABLESPACE UNLOAD ONLY or EXTERNAL	Yes	Yes	Yes
REPAIR DUMP or VERIFY	Yes	Yes	Yes

Table 138. Compatibility of UNLOAD with other utilities (continued)

Action	UNLOAD SHRLEVEL REFERENCE	UNLOAD SHRLEVEL CHANGE	FROM IMAGE COPY
REPAIR LOCATE INDEX PAGE REPLACE	Yes	Yes	Yes
REPAIR LOCATE KEY or RID DELETE or REPLACE	No	No	Yes
REPAIR LOCATE TABLESPACE PAGE REPLACE	No	No	Yes
REPORT	Yes	Yes	Yes
RUNSTATS INDEX	Yes	Yes	Yes
RUNSTATS TABLESPACE	Yes	Yes	Yes
STOSPACE	Yes	Yes	Yes

Note (*): If the same data set is used as the output from the COPY utility and as the input data set of the UNLOAD utility, unexpected results can occur.

Sample UNLOAD control statements

Example 1: Unloading all columns of specified rows. The control statement in Figure 113 specifies that all columns of rows that meet the following criteria are to be unloaded from table DSN8810.EMP in table space DSN8D81A.DSN8S71E:

- The value in the WORKDEPT column is D11.
- The value in the SALARY column is greater than 25 000.

```
//STEP1    EXEC DSNUPROC,UID='SMPLUNLD',UTPROC='',SYSTEM='DSN'
//SYSREC   DD DSN=USERID.SMPLUNLD.SYSREC,
//         DISP=(NEW,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(TRK,(2,1))
//SYSPUNCH DD DSN=USERID.SMPLUNLD.SYSPUNCH,
//         DISP=(NEW,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(TRK,(1,1))
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  UNLOAD TABLESPACE DSN8D81A.DSN8S81E
    FROM TABLE DSN8810.EMP
    WHEN (WORKDEPT = 'D11' AND SALARY > 25000)
```

Figure 113. Example of unloading all columns of specified rows

Example 2: Unloading specific columns by using a field specification list. The following control statement specifies that columns EMPNO, LASTNAME, and SALARY are to be unloaded, in that order, for all rows that meet the specified conditions. These conditions are specified in the WHEN clause and are the same as those conditions in example 1. The SALARY column is to be unloaded as type DECIMAL EXTERNAL. The NOPAD option indicates that variable-length fields are to be unloaded without any padding.

```
UNLOAD TABLESPACE DSN8D81A.DSN8S81E NOPAD
  FROM TABLE DSN8810.EMP
    (EMPNO, LASTNAME, SALARY DECIMAL EXTERNAL)
  WHEN (WORKDEPT = 'D11' AND SALARY > 25000)
```

The output from this example might look similar to the following output:

```
000060@@STERN#   32250.00
000150@@ADAMSON#   25280.00
000200@@BROWN#   27740.00
000220@@LUTZ#   29840.00
200220@@JOHN#   29840.00
```

In this output:

- '@@' before the last name represents the 2-byte binary field that contains the length of the VARCHAR field LASTNAME (for example, X'0005' for STERN).
- '#' represents the NULL indicator byte for the nullable SALARY field.
- Because the SALARY column is declared as DECIMAL (9,2) on the table, the default output length of the SALARY field is 11 (9 digits + sign + decimal point), not including the NULL indicator byte.
- LASTNAME is unloaded as a variable-length field because the NOPAD option is specified.

Example 3: Unloading data from an image copy. The FROMCOPY option in the following control statement specifies that data is to be unloaded from a single image copy data set, JUKWU111.FCOPY1.STEP1.FCOPY1.

PUNCHDDN SYSPUNCH specifies that the UNLOAD utility is to generate LOAD utility control statements and write them to the data set that is defined by the SYSPUNCH DD statement; SYSPUNCH is the default. UNLDDN SYSREC specifies that the data is to be unloaded to the data set that is defined by the SYSREC DD statement; SYSREC is the default.

```
UNLOAD TABLESPACE DBKW1101.TPKW1101
           FROMCOPY JUKWU111.FCOPY1.STEP1.FCOPY1
           PUNCHDDN SYSPUNCH UNLDDN SYSREC
```

Example 4: Unloading a sample of rows and specifying a header. The following control statement specifies that a sample of rows is to be unloaded from table ADMF001.TBKW1605. Unloading a sample of rows is useful for building a test system. The SAMPLE option indicates that 75% of the rows are to be sampled. The HEADER option indicates that the string 'sample' is to be used as the header field in the output file. The PUNCHDDN option indicates that UNLOAD is to generate LOAD utility control statements and write them to the SYSPUNCH data set, which is the default. UNLOAD specifies the header field as a criterion in the WHEN clause of these LOAD statements.

```
UNLOAD TABLESPACE DBKW1603.TPKW1603
            PUNCHDDN SYSPUNCH UNLDDN SYSREC
            FROM TABLE ADMF001.TBKW1605
              HEADER CONST 'sample'
              SAMPLE 75
```

Example 5: Unloading data from two tables in a segmented table space. The following control statement specifies that data from table ADMF001.TBKW1504 and table ADMF001.TBKW1505 is to be unloaded from the segmented table space DBKW1502.TSKW1502. The PUNCHDDN option indicates that UNLOAD is to generate LOAD utility control statements and write them to the SYSPUNCH data

set, which is the default. The UNLDDN option specifies that the data is to be unloaded to the data set that is defined by the SYSREC DD statement, which is also the default.

```
UNLOAD TABLESPACE DBKW1502.TSKW1502
           PUNCHDDN SYSPUNCH UNLDDN SYSREC
            FROM TABLE ADMF001.TBKW1504
            FROM TABLE ADMF001.TBKW1505
```

Example 6: Unloading data in parallel from a partitioned table space. The UNLOAD control statement in Figure 114 specifies that data from table TCRT.TTBL is to be unloaded to data sets that are defined by the UNLDDS template. These data sets are to be dynamically allocated and named according to the naming convention that is defined by the DSN option of the TEMPLATE utility control statement. This naming convention indicates that a data set is to be allocated for each table space partition. For more information about TEMPLATE control statements, see "Syntax and options of the TEMPLATE control statement " on page 593 in the TEMPLATE chapter.

Assume that table space TDB1.TSP1, which contains table TCRT.TTBL, has three partitions. Because the table space is partitioned and each partition is associated with an output data set that is defined by the UNLDDS template, the UNLOAD job runs in parallel in a multi-processor environment. The number of parallel tasks are determined by the number of available processors.

```
//STEP1    EXEC DSNUPROC,UID='SMPLUNLD',UTPROC='',SYSTEM='DSN'
//SYSPUNCH DD DSN=USERID.SMPLUNLD.SYSPUNCH,
//         DISP=(NEW,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(TRK,(1,1))
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  TEMPLATE UNLDDS   DSN &USERID..SMPLUNLD.&TS..P&PART.
           UNIT SYSDA  DISP (NEW,CATLG,CATLG) SPACE (2,1) CYL
  UNLOAD TABLESPACE TDB1.TSP1
    UNLDDN UNLDDS
    FROM TABLE TCRT.TTBL
```

Figure 114. Example of unloading data in parallel from a partitioned table space

Assume that the user ID is USERID. This UNLOAD job creates the following three data sets to store the unloaded data:

- USERID.SMPLUNLD.TSP1.P00001 ... contains rows from partition 1.
- USERID.SMPLUNLD.TSP1.P00002 ... contains rows from partition 2.
- USERID.SMPLUNLD.TSP1.P00003 ... contains rows from partition 3.

Example 7: Using a LISTDEF utility statement to specify partitions to unload. The UNLOAD control statement in Figure 115 on page 665 specifies that data that is included in the UNLDLIST list is to be unloaded. UNLDLIST is defined in the LISTDEF utility control statement and contains partitions one and three of table space TDB1.TSP1. The LIST option of the UNLOAD statement specifies that the UNLOAD utility is to use this list. For more information about LISTDEF control statements, see "Syntax and options of the LISTDEF control statement" on page 173 in the LISTDEF chapter.

The data is to be unloaded to data sets that are defined by the UNLDDS template. For more information about TEMPLATE control statements, see "Syntax and options of the TEMPLATE control statement " on page 593 in the TEMPLATE chapter.

```
//SAMPJOB  JOB ...
//STEP1    EXEC DSNUPROC,UID='SMPLUNLD',UTPROC='',SYSTEM='DSN'
//SYSPUNCH DD DSN=USERID.SMPLUNLD.SYSPUNCH,
//         DISP=(NEW,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(TRK,(1,1))
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  LISTDEF  UNLDLIST
           INCLUDE  TABLESPACE TDB1.TSP1 PARTLEVEL(1)
           INCLUDE  TABLESPACE TDB1.TSP1 PARTLEVEL(3)
  TEMPLATE UNLDDS   DSN &USERID..SMPLUNLD.&TS..P&PART.
           UNIT SYSDA  DISP (NEW,CATLG,CATLG) SPACE (2,1) CYL
  UNLOAD LIST UNLDLIST              -- LIST name
    UNLDDN UNLDDS                   -- TEMPLATE name
```

Figure 115. Example of using a LISTDEF utility statement to specify partitions to unload

Assume that the user ID is USERID. This UNLOAD job creates the following two data sets to store the unloaded data:
- USERID.SMPLUNLD.TSP1.P00001 ... contains rows from partition 1.
- USERID.SMPLUNLD.TSP1.P00003 ... contains rows from partition 3.

Example 8: Unloading multiple table spaces by using LISTDEF. The UNLOAD control statement in Figure 116 specifies that data from multiple table spaces is to be unloaded. These table spaces are specified in the LISTDEF utility control statement. Assume that the database TDB1 contains two table spaces that can be expressed by the pattern-matching string 'TSP*', (for example, TSP1 and TSP2). These table spaces are both included in the list named UNLDLIST, which is defined in the LISTDEF statement. The LIST option of the UNLOAD statement specifies that the UNLOAD utility is to use this list. For more information about LISTDEF control statements, see “Syntax and options of the LISTDEF control statement” on page 173 in the LISTDEF chapter.

The UNLDDN option specifies that the data is to be unloaded to data sets that are defined by the UNLDDS template. The PUNCHDDN option specifies that UNLOAD is to generate LOAD utility control statements and write them to the data sets that are defined by the PUNCHDS template. For more information about TEMPLATE control statements, see “Syntax and options of the TEMPLATE control statement ” on page 593 in the TEMPLATE chapter.

```
//SAMPJOB  JOB ...
//STEP1    EXEC DSNUPROC,UID='SMPLUNLD',UTPROC='',SYSTEM='DSN'
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  LISTDEF  UNLDLIST
           INCLUDE  TABLESPACE TDB1.TSP*
  TEMPLATE UNLDDS   DSN &USERID..SMPLUNLD.&TS.
           UNIT SYSDA  DISP (NEW,CATLG,CATLG) SPACE (2,1) CYL
  TEMPLATE PUNCHDS  DSN &USERID..SMPLPUNC.&TS.
           UNIT SYSDA  DISP (NEW,CATLG,CATLG) SPACE (1,1) CYL
  UNLOAD LIST UNLDLIST
    PUNCHDDN PUNCHDS             -- TEMPLATE name
    UNLDDN   UNLDDS              -- TEMPLATE name
```

Figure 116. Example of unloading multiple table spaces

Assume that the user ID is USERID. This UNLOAD job creates the following two data sets to store the unloaded data:
- USERID.SMPLUNLD.TSP1 ... contains rows from table space TDB1.TSP1.
- USERID.SMPLUNLD.TSP2 ... contains rows from table space TDB1.TSP2.

Example 9: Unloading data into a delimited file. The control statement in Figure 117 specifies that data from the specified columns (RECID, CHAR7SBCS, CHAR7BIT, VCHAR20, VCHAR20SBCS, VCHAR20BIT) in table TBQB0501 is to be unloaded into a delimited file. This output format is indicated by the DELIMITED option. The POSITION(*) option indicates that each field in the output file is to start at the first byte after the last position of the previous field.

The column delimiter is specified by the COLDEL option as a semicolon (;), the character string delimiter is specified by the CHARDEL option as a pound sign (#), and the decimal point character is specified by the DECPT option as an exclamation point (!).

PUNCHDDN SYSPUNCH specifies that UNLOAD is to generate LOAD utility control statements and store them in the SYSPUNCH data set, which is the default. UNLDDN SYSREC indicates that the data is to be unloaded to the SYSREC data set, which is the default.

The EBCDIC option indicates that all output character data is to be in EBCDIC.

```
//*
//STEP3    EXEC DSNUPROC,UID='JUQBU105.UNLD1',
//         UTPROC='',
//         SYSTEM='SSTR'
//UTPRINT  DD SYSOUT=*
//SYSREC   DD DSN=JUQBU105.UNLD1.STEP3.TBQB0501,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSPUNCH DD DSN=JUQBU105.UNLD1.STEP3.SYSPUNCH
//         DISP=(MOD,CATLG,CATLG)
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD*
  UNLOAD TABLESPACE DBQB0501.TSQB0501
         DELIMITED CHARDEL '#' COLDEL ';' DECPT '!'
         PUNCHDDN SYSPUNCH
         UNLDDN SYSREC EBCDIC
         FROM TABLE ADMF001.TBQB0501
          (RECID       POSITION(*) CHAR,
           CHAR7SBCS   POSITION(*) CHAR,
           CHAR7SBIT   POSITION(*) CHAR(7),
           VCHAR20     POSITION(*) VARCHAR,
           VCHAR20SBCS POSITION(*) VARCHAR,
           VCHAR20BIT  POSITION(*) VARCHAR)
 /*
```

Figure 117. Example of unloading data into a delimited file.

Example 10: Converting character data. For this example, assume that table DSN8810.DEMO_UNICODE contains character data in Unicode. The UNLOAD control statement in Figure 118specifies that the utility is to unload the data in this table as EBCDIC data.

```
UNLOAD
  EBCDIC
  TABLSPACE DSN8D81E.DSN8S81U
  FROM TABLE DSN8810.DEMO_UNICODE
```

Figure 118. Example of unloading Unicode table data into EBCDIC

Example 11: Unloading LOB data to a file. The UNLOAD control statement in Figure 119 on page 667 specifies that the utility is to unload data from table DSN8910.EMP_PHOTO_RESUME into the data set that is identified by the

SYSREC DD statement. Data in the EMPNO field is six bytes of character data, as indicated by the CHAR(6) option, and is unloaded directly into the SYSREC data set. Data in the RESUME column is CLOB data as indicated by the CLOBF option. This CLOB data is to be unloaded to the files identified by the LOBFRV template, which is defined in the preceding TEMPLATE statement. If these files do not already exist, DB2 creates them. The names of these files are stored in the SYSREC data set. The length of the file name to be stored in this data set can be up to 255 bytes as specified by the VARCHAR option.

```
TEMPLATE LOBFRV DSN 'UNLDTEST.&DB..&TS..RESUME'
                DSNTYPE(PDS) UNIT(SYSDA)

UNLOAD DATA
  FROM TABLE DSN8910.EMP_PHOTO_RESUME
  (EMPNO  CHAR(6),
   RESUME VARCHAR(255) CLOBF LOBFRV)
 SHRLEVEL CHANGE
```

Figure 119. Example of unloading LOB data into a file

Part 3. DB2 stand-alone utilities

Chapter 33. Invoking stand-alone utilities

This chapter contains procedures and guidelines for creating utility control statements and EXEC PARM parameters for invoking the stand-alone utilities.

Utility control statements and parameters define the function that a utility job performs. Some stand-alone utilities read the control statements from an input stream, and others obtain the function definitions from JCL EXEC PARM parameters.

The following topics provide additional information:

- "Creating utility control statements"
- "Specifying options by using the JCL EXEC PARM parameter"
- "Effects of invoking stand-alone utilities on tables that have multilevel security with row-level granularity" on page 672

Creating utility control statements

Create the utility control statements with the ISPF/PDF edit function. After you create the control statements, save them in a sequential or partitioned data set.

The following utilities read control statements from the input stream file of the specified DD name:

Utility	DD name
DSNJU003 (change log inventory)	SYSIN
DSNJU004 (print log map)	SYSIN (optional)
DSN1LOGP	SYSIN
DSN1SDMP	SDMPIN

Utility control statements are read from the DD name input stream. The statements in that stream must conform to the following rules:

- The logical record length (LRECL) must be 80 characters. Columns 73 through 80 are ignored.
- The records are concatenated into a single stream before they are parsed. No concatenation character is necessary.
- The SYSIN stream can contain multiple utility control statements.

Specifying options by using the JCL EXEC PARM parameter

Use the EXEC PARM parameter to specify function options for the following stand-alone utilities:

- DSN1CHKR
- DSN1COMP
- DSN1COPY
- DSN1PRNT

Ensure that the parameters that you specify obey the following OS/390 JCL EXEC PARM parameter specification rules:

- Enclose multiple subparameters in single quotation marks or parentheses and separate the subparameters with commas, as in the following example:

  ```
  //name   EXEC PARM='ABC,...,XYZ'
  ```

- Do not let the total length exceed 100 characters.
- Do not use blanks within the parameter specification.

To specify the parameter across multiple lines, perform the following actions:

1. Enclose it in parentheses.
2. End the first line with a subparameter, followed by a comma.
3. Continue the subparameters on the next line, beginning before column 17.

The following example shows a parameter that spans multiple lines:

```
//stepname  EXEC PARM=(ABC,...LMN,
           OPQ,...,XYZ)
```

Effects of invoking stand-alone utilities on tables that have multilevel security with row-level granularity

If you use RACF access control with multilevel security, you do not need any additional authorizations to run stand-alone utilities. When processing tables that have multilevel security with row-level granularity, stand-alone utilities ignore row-level granularity. They check only for authorization to operate on the table space; they do not check row-level authorizations. For more information about multilevel security, see Part 3 of *DB2 Administration Guide*.

Chapter 34. DSNJCNVB

The DSNJCNVB 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 1 000 archive log volumes and 31 active log data sets per log copy. Converting the BSDS is optional.

The following topics provide additional information:

- "Before running DSNJCNVB"
- "Running DSNJCNVB" on page 674
- "Sample DSNJCNVB control statement" on page 674
- "DSNJCNVB output " on page 674

Before running DSNJCNVB

This section contains information that you need to be aware of prior to running DSNJCNVB.

Environment

Execute the DSNJCNVB utility as a batch job only when DB2 is not running.

Your DB2 subsystem must be in new-function mode to convert the BSDS.

Authorization required

The authorization ID of the DSNJCNVB job must have the requisite RACF authorization.

Prerequisite actions

If you have migrated to a new version of DB2, you need to create a larger BSDS before converting it. See the *DB2 Installation Guide* for instructions on how to create a larger BSDS. For a new installation, you do not need to create a larger BSDS. DB2 provides a larger BSDS definition in installation job DSNTIJIN; however, if you want to convert the BSDS, you must still run DSNJCNVB.

Control statement

See "Sample DSNJCNVB control statement" on page 674 for an example of using DSNJCNVB to convert the BSDS.

Required and optional data sets

DSNJCNVB recognizes DD statements with the following DD names:

JOBCAT or STEPCAT Specifies the catalog in which the BSDS is cataloged. This statement is optional. Typically, the high-level qualifier of the BSDS name points to the ICF catalog that contains an entry for the BSDS.

SYSUT1 Specifies the BSDS copy 1 data set that DSNJCNVB is to use as input. This statement is required.

SYSUT2 Specifies the BSDS copy 2 data set that DSNJCNVB is to use as input. This statement is optional.

Specify this statement if you are using dual BSDSs and you want to convert both with a single execution of DSNJCNVB. You can run DSNJCNVB separately for each copy if desired.

SYSPRINT Specifies a data set or print spool class for print output. This statement is required. The logical record length (LRECL) is 125.

Running DSNJCNVB

Use the following EXEC statement to execute this utility:

```
//EXEC   PGM=DSNJCNVB
```

Sample DSNJCNVB control statement

The following statements specify that DSNJCNVB is to convert the BSDS so that it can manage up to 10 000 archive log volumes and 93 active log data sets per log copy. The SYSUT1 and SYSUT2 statements identify the bootstrap data sets. Only the SYSUT1 statement is required. The SYSUT2 statement is optional. Specify SYSUT2 only if you are using dual BSDSs and you want to convert both with a single execution of DSNJCNVB.

```
//DSNJCNVB EXEC PGM=DSNJCNVB
//STEPLIB  DD  DISP=SHR,DSN=DSNC810.SDSNEXIT
//         DD  DISP=SHR,DSN=DSNC810.SDSNLOAD
//SYSUT1   DD  DISP=OLD,DSN=DSNC810.BSDS01
//SYSUT2   DD  DISP=OLD,DSN=DSNC810.BSDS02
//SYSPRINT DD  SYSOUT=*
```

DSNJCNVB output

The following example shows sample DSNJCNVB output:

```
CONVERSION OF BSDS DATA SET - COPY 1, DSN=DSNC810.BSDS01
         SYSTEM TIMESTAMP   - DATE=2003.199  LTIME= 9:40:58.74
         UTILITY TIMESTAMP  - DATE=2003.216  LTIME=14:26:02.21
         PREVIOUS HIKEY     - 04000053
         NEW HIKEY          - 040002F0
         RECORDS ADDED      - 669
DSNJ260I  DSNJCNVB BSDS CONVERSION FOR DDNAME=SYSUT1 COMPLETED SUCCESSFULLY
DSNJ200I  DSNJCNVB CONVERT BSDS UTILITY PROCESSING COMPLETED SUCCESSFULLY
```

Chapter 35. DSNJLOGF (preformat active log)

When writing to an active log data set for the first time, DB2 must preformat a VSAM control area before writing the log records. The DSNJLOGF utility avoids this delay by preformatting the active log data sets before bringing them online to DB2.

The following topics provide additional information:

- "Before running DSNJLOGF"
- "Sample DSNJLOGF control statement"
- "DSNJLOGF output" on page 676

Before running DSNJLOGF

This section contains information that you need to be aware of prior to running DSNJLOGF.

Environment

Run DSNJLOGF as a z/OS job.

Control statement

See "Sample DSNJLOGF control statement" for an example of using DSNJLOGF to preformat the active log data sets.

Required data sets: DSNJLOGF recognizes DD statements with the following DD names.

SYSUT1 Defines the newly defined active log data set that is to be preformatted. The data set must be an empty VSAM linear data set and less than four gigabytes in size.

SYSPRINT Defines the print spool class or data set for print output. The logical record length (LRECL) is 132.

Sample DSNJLOGF control statement

The control statements in Figure 120 on page 676 specify that DSNJLOGF is to preformat the four active log data sets that are identified by the four SYSUT1 DD statements.

```
//JOBLIB DD  DSN=DSN810.SDSNLOAD,DISP=SHR
//STEP1    EXEC PGM=DSNJLOGF
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSUT1   DD DSN=DSNC810.LOGCOPY1.DS01,DISP=SHR
//STEP2    EXEC PGM=DSNJLOGF
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSUT1   DD DSN=DSNC810.LOGCOPY1.DS02,DISP=SHR
//STEP3    EXEC PGM=DSNJLOGF
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSUT1   DD DSN=DSNC810.LOGCOPY2.DS01,DISP=SHR
//STEP4    EXEC PGM=DSNJLOGF
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSUT1   DD DSN=DSNC810.LOGCOPY2.DS02,DISP=SHR
```

Figure 120. Sample DSNJLOGF control statement

DSNJLOGF output

The following sample shows the DSNJLOGF output for the first data set in the sample control statement in Figure 120.

```
DSNJ991I DSNJLOGF START OF LOG DATASET PREFORMAT FOR JOB LOGFRMT  STEP1
DSNJ992I DSNJLOGF LOG DATA SET NAME = DSNC810.LOGCOPY1.DS01
DSNJ996I DSNJLOGF LOG PREFORMAT COMPLETED SUCCESSFULLY, 00015000
         RECORDS FORMATTED
```

Chapter 36. DSNJU003 (change log inventory)

The DSNJU003 stand-alone utility changes the bootstrap data sets (BSDSs). You can use the utility to:

- Add or delete active or archive log data sets
- Add or delete checkpoint records
- Create a conditional restart control record to control the next start of the DB2 subsystem
- Change the VSAM catalog name entry in the BSDS
- Modify the communication record in the BSDS
- Modify the value for the highest-written log RBA value (relative byte address within the log) or the highest-offloaded RBA value

The following topics provide additional information:

- "Syntax and options of the DSNJU003 control statement"
- "Before running DSNJU003" on page 687
- "Using DSNJU003 to modify the BSDS" on page 689
- "Sample DSNJU003 control statements" on page 694

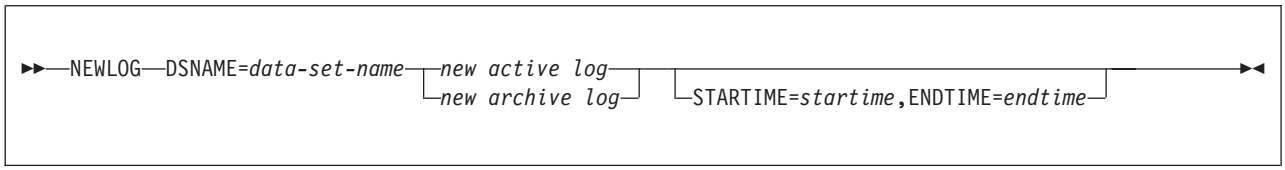
Syntax and options of the DSNJU003 control statement

DSNJU003 (change log inventory) syntax diagram

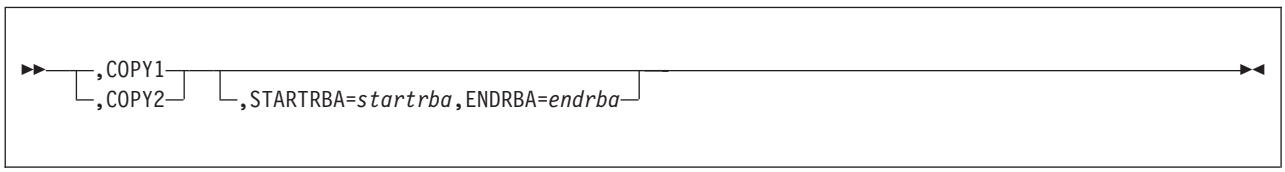
DSNJU003 uses multiple statements that you submit in separate jobs. The statements are:

- NEWLOG
- DELETE
- CRESTART
- NEWCAT
- DDF
- CHECKPT
- HIGHRBA

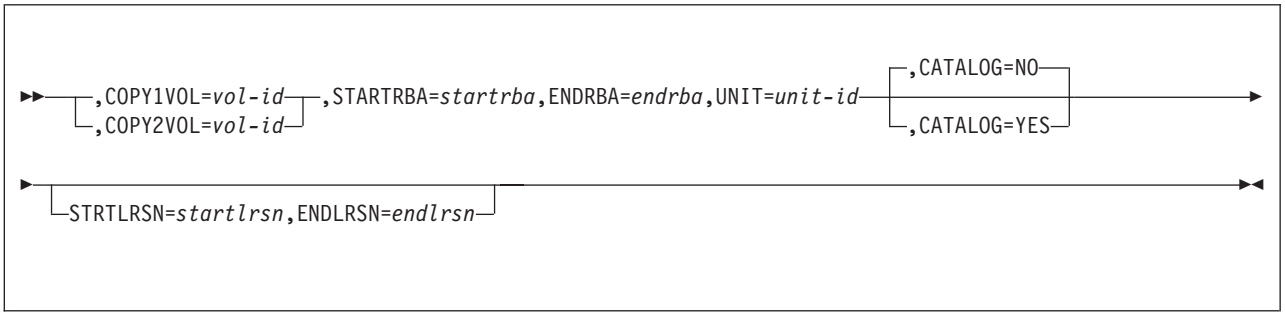
NEWLOG statement



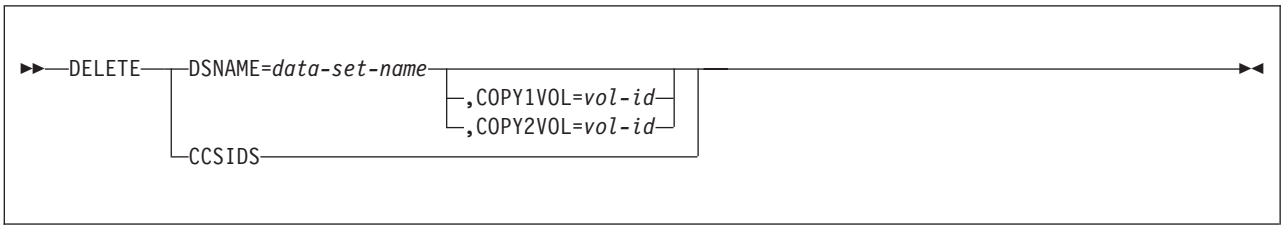
new active log:



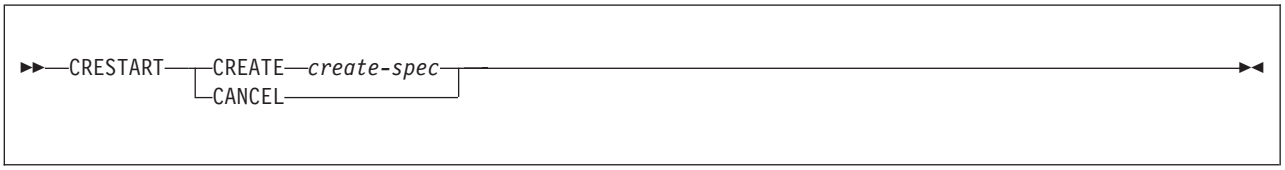
new archive log:



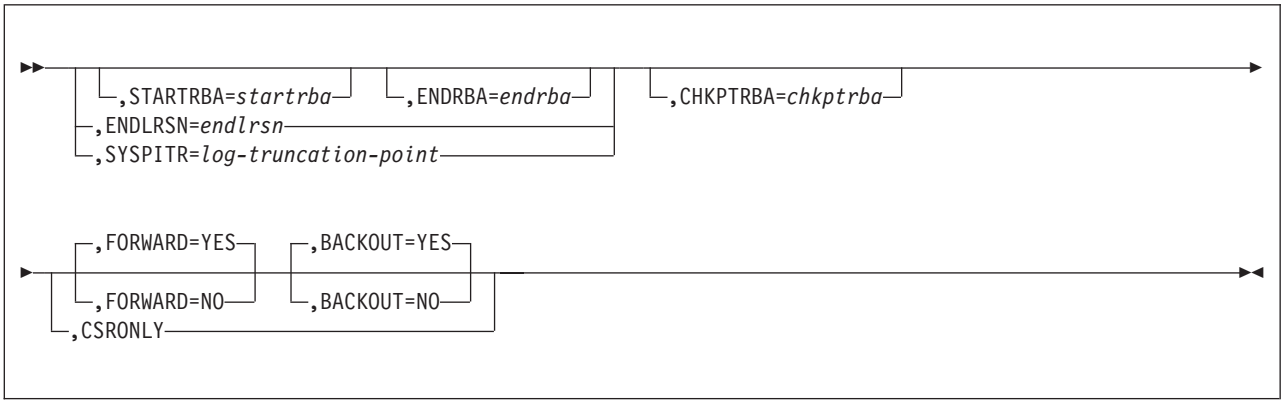
DELETE statement



CRESTART statement

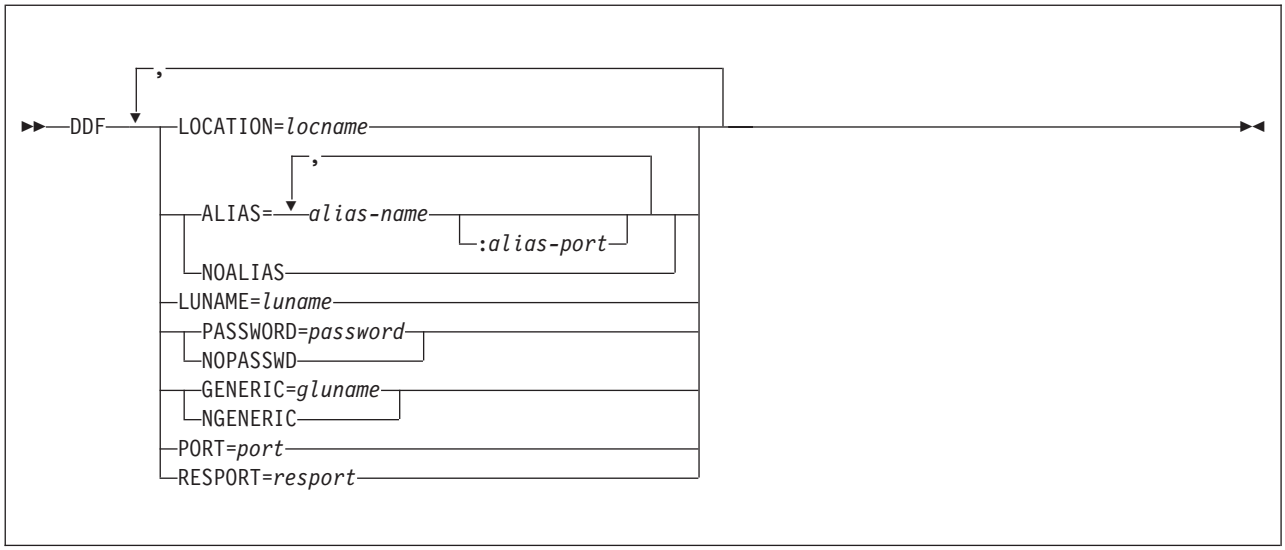


create-spec:

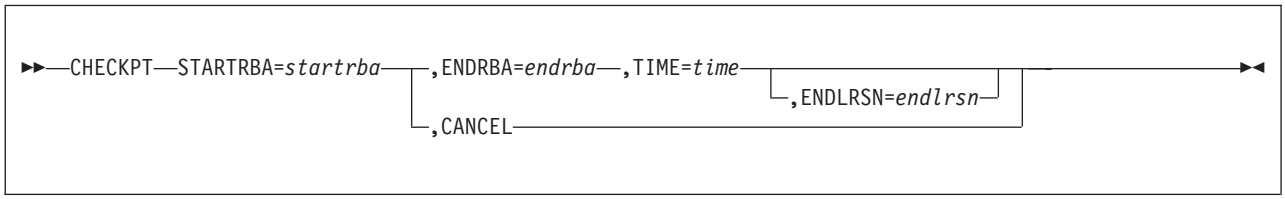


NEWCAT statement

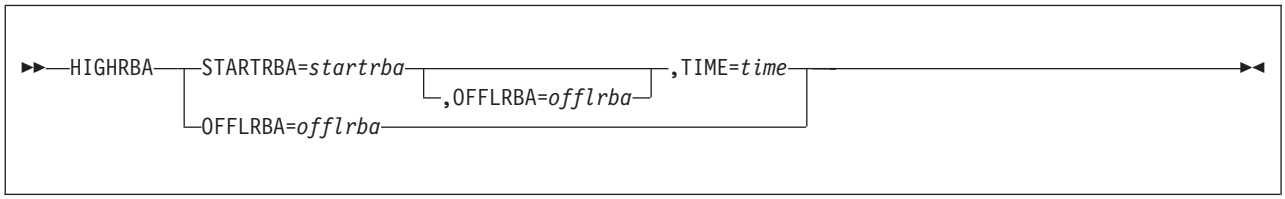
DDF statement



CHECKPT statement



HIGHRBA statement



Option descriptions

"Creating utility control statements" on page 671 provides general information about specifying options for DB2 utilities.

NEWLOG Declares one of the following data sets:

- A VSAM data set that is available for use as an active log data set.

Use only the keywords DSNAME=, COPY1, and COPY2.

- An active log data set that is replacing one that encountered an I/O error.

  Use only the keywords DSNAME=, COPY1, COPY2, STARTRBA=, and ENDRBA=.

- An archive log data set volume.

  Use only the keywords DSNAME= ,COPY1VOL=, COPY2VOL=, STARTRBA=, ENDRBA=, UNIT=, CATALOG=, STRTLRSN=, and ENDLRSN=.

  If you create an archive log data set and add it to the BSDS with this utility, you can specify a name that DB2 might also generate. DB2 generates archive log data set names of the form DSNCAT.ARCHLOG*x*.A*nnnnnnn* where:

  – DSNCAT and ARCHLOG are parts of the data set prefix that you specified on installation panels DSNTIPA2 and DSNTIPH.
  – *x* is 1 for the first copy of the logs, and 2 is for the second copy.
  – A*nnnnnnn* represents the series of low-level qualifiers that DB2 generates for archive log data set names, beginning with A0000001, and incrementing to A0000002, A0000003, and so forth.

  For data sharing, the naming convention is DSNCAT.ARCHLOG1 or DSNCAT.DSN1.ARCLG1.

  If you do specify a name by using the same naming convention as DB2, you receive a dynamic allocation error when DB2 generates that name. The error message, DSNJ103I, is issued once. DB2 then increments the low-level qualifier to generate the next data set name in the series and offloads to it the next time DB2 archives. (The active log that previously was not offloaded is offloaded to this data set.)

  The newly defined active logs cannot specify a start and end LRSN. When DB2 starts, it reads the new active log data sets with an RBA range to determine the LRSN range, and updates the start and end LRSN in the BSDS for the new log data sets. The start and end LRSN for new active logs that contain active log data are read at DB2 start-up time from the new active log data sets that are specified in the change log inventory NEWLOG statements. For new archive logs that are defined with change log inventory, the user must specify the start and end RBAs. For data sharing, the user must also specify the start and end LRSNs. DB2 startup does not attempt to find these values from the new archive log data sets.

DSNAME=*data-set-name*
Specifies a log data set.

data-set-name can be up to 44 characters long.

COPY1 Makes the data set an active log copy-1 data set.

COPY2 Makes the data set an active log copy-2 data set.

STARTRBA=*startrba*
Identifies a hexadecimal number of up to 12 characters. If you use fewer than 12 characters, leading zeros are added. *startrba* must

end with '000'; otherwise DB2 returns a DSNJ4381 error message. You can obtain the RBA from messages or by printing the log map.

On the NEWLOG statement, *startrba* gives the log RBA of the beginning of the replacement active log data set or the archive log data set volume that is specified by DSNAME.

On the CRESTART statement, *startrba* is the earliest RBA of the log that is to be used during restart. If you omit STARTRBA, DB2 determines the beginning of the log range.

On the CHECKPT statement, *startrba* indicates the start checkpoint log record.

STARTRBA is required when STARTIME is specified.

On the HIGHRBA statement, *startrba* denotes the log RBA of the highest-written log record in the active log data sets.

ENDRBA=*endrba*

endrba is a hexadecimal number of up to 12 characters. If you use fewer than 12 characters, leading zeros are added. *endrba* must end with '000' or DB2 returns a DSNJ4381 error message.

On the NEWLOG statement, *endrba* gives the log RBA (relative byte address within the log) of the end of the replacement active log data set or the archive log data set volume that is specified by DSNAME.

On the CRESTART statement, *endrba* is the last RBA of the log that is to be used during restart, and it is also the starting RBA of the next active log that is written after restart. Any log information in the bootstrap data set, the active logs, and the archive logs with an RBA that is greater than *endrba* is discarded. If you omit ENDRBA, DB2 determines the end of the log range.

The value of ENDRBA must be a multiple of 4096. (The hexadecimal value must end in 000.) Also, the value must be greater than or equal to the value of STARTRBA. If STARTRBA and ENDRBA are equal, the next restart is a cold start; that is, no log records are processed during restart. The specified RBA becomes the beginning RBA of the new log.

On the CHECKPT statement, *endrba* indicates the end checkpoint log record that corresponds to the start checkpoint log record.

COPY1VOL=*vol-id*

vol-id is the volume serial of the copy-1 archive log data set that is specified after DSNAME.

COPY2VOL=*vol-id*

vol-id is the volume serial of the copy-2 archive log data set that is specified after DSNAME.

UNIT=*unit-id* *unit-id* is the device type of the archive log data set that is named after DSNAME.

CATALOG Indicates whether the archive log data set is to be cataloged.

NO Indicates that the archive log data set is not to be cataloged. All subsequent allocations of the data set are made using the unit and volume information that is specified on the statement.

YES Indicates that the archive log data set is to be cataloged. All subsequent allocations of the data set are made using the catalog.

DB2 requires that all archive log data sets on disk be cataloged. Select CATALOG=YES if the archive log data set is on disk.

STRTLRSN=*startlrsn*

On the NEWLOG statement, identifies the LRSN in the log record header of the first complete log record on the new archive data set. *startlrsn* is a hexadecimal number of up to 12 characters. If you use fewer than 12 characters, leading zeros are added. In a data sharing environment, run the print log map utility to find an archive log data set and start and end RBAs and LRSNs.

ENDLRSN=*endlrsn*

endlrsn is a hexadecimal number of up to 12 characters. If you use fewer than 12 characters, leading zeros are added. In a data sharing environment, run the print log map utility to find an archive log data set and start and end RBAs and LRSNs.

For the NEWLOG and CHECKPT statements, the ENDLRSN option is valid only in a data sharing environment. For the CRESTART statement, the ENDLRSN option is valid in both data sharing and non-data sharing environments. This option cannot be specified with STARTRBA or ENDRBA.

On the NEWLOG statement, *endlrsn* is the LRSN in the log record header of the last log record on the new archive data set.

On the CRESTART statement, in a data sharing environment, *endlrsn* is an LRSN value that is to be used as the log truncation point. A valid log truncation point is any LRSN value for which there exists a log record with an LRSN that is greater than or equal to the specified LRSN value. Any log information in the bootstrap data set, the active logs, and the archive logs with an LRSN greater than *endlrsn* is discarded. If you omit ENDLRSN, DB2 determines the end of the log range.

In a non-data sharing environment, *endlrsn* is 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 *endlrsn* is discarded. If the *endlrsn* RBA value does not match the start of a log record, DB2 restart fails. If you omit ENDLRSN, DB2 determines the end of the log range.

On the CHECKPT statement, *endlrsn* is the LRSN of the end checkpoint log record.

STARTIME=*startime*

Enables you to record the start time of the RBA in the BSDS. This field is optional.

startime specifies the start time in the following timestamp format:

```
yyyydddhhmmsst
```

In this format:

yyyy Indicates the year (1989-2099).
ddd Indicates the day of the year (0-365; 366 in leap years).

hh Indicates the hour (0-23).
mm Indicates the minutes (0-59).
ss Indicates the seconds (0-59).
t Indicates tenths of a second.

If fewer than 14 digits are specified for the STARTIME or ENDTIME parameter, trailing zeros are added.

If STARTIME is specified, the ENDTIME, STARTRBA, and ENDRBA options must also be specified.

ENDTIME=*endtime*
Enables you to record the end time of the RBA in the BSDS. This field is optional.

endtime specifies the end time in the same timestamp format as the STARTIME option. The ENDTIME value must be greater than or equal to the value of STARTIME.

DELETE Deletes either CCSID information or log data set information from the bootstrap data sets. To delete CCSID information, specify the CCSIDS option. To delete all information for a specified log data set or volume, specify the DSNAME option.

CCSIDS Deletes CCSID information from 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 CCSID values from DSNHDECP are recorded in the BSDS the next time DB2 is started.

CRESTART Controls the next restart of DB2, either by creating a new conditional restart control record or by canceling the one that is currently active.

CREATE Creates a new conditional restart control record. When the new record is created, the previous control record becomes inactive.

SYSPITR=*log-truncation-point*
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 for system recovery. Before you run the RESTORE SYSTEM utility to recover system data, you must use the SYSPITR option of DSNJU003. This option enables you to create a conditional restart control record to truncate the logs for system point-in-time recovery.

log-truncation-point specifies the log RBA or log LRSN. In a non-data sharing environment, *log-truncation point* is the RBA value that matches the start of the last log record that is to be used during restart. If the RBA value does not match the start of a log record, DB2 restart fails. In a data sharing environment, *log-truncation point* is an LRSN value that is a valid log truncation point. A valid log truncation point is any LRSN value for which there exists a log record with an LRSN that is greater than or equal to the specified LRSN value. Use the same LRSN value for all members of the data sharing group that require log truncation.

| You cannot specify any other option with CREATE, SYSPITR. You
| can run this option of the utility only after new-function mode is
| enabled.

CANCEL **On the CRESTART statement,** deactivates the currently active conditional restart control record. The record remains in the BSDS as historical information.

No other keyword can be used with CANCEL on the CRESTART statement.

On the CHECKPT statement, deletes the checkpoint queue entry that contains a starting RBA that matches the parameter that is specified by the STARTRBA keyword.

Attention: This statement can override DB2's efforts to maintain data in a consistent state. Do not use this statement without understanding the conditional restart process, which is described in Part 4 (Volume 1) of *DB2 Administration Guide*.

CHKPTRBA=*chkptrba*
Identifies the log RBA of the start of the checkpoint record that is to be used during restart.

If you use STARTRBA or ENDRBA, and you do not use CHKPTRBA, the DSNJU003 utility selects the RBA of an appropriate checkpoint record. If you do use CHKPTRBA, you override the value that is selected by the utility.

chkptrba must be in the range that is determined by *startrba* and *endrba* or their default values.

If possible, do not use CHKPTRBA; let the utility determine the RBA of the checkpoint record.

CHKPTRBA=0 overrides any selection by the utility; at restart, DB2 attempts to use the most recent checkpoint record.

FORWARD= Indicates whether to use the forward-log-recovery phase of DB2 restart, which reads the log in a forward direction to recover any units of recovery that were in one of the following two states when DB2 was last stopped:

- Indoubt (the units of recovery had finished the first phase of commit, but had not started the second phase)
- In-commit (had started but had not finished the second phase of commit)

For a complete description of the forward-log-recovery phase, see Part 4 of *DB2 Administration Guide*.

YES Allows forward-log recovery.

If you specify a cold start (by using the same value for STARTRBA and ENDRBA), no recovery processing is performed.

NO Terminates forward-log recovery before log records are
processed. When you specify, FORWARD=NO, DB2 does
not go back in the log to the beginning of any indoubt or
in-commit units of recovery to complete forward recovery
for these units. Choose this option if a very old indoubt
unit of recovery exists to avoid a lengthy restart. The

in-commit and indoubt units of recovery are marked as bypassed and complete in the log. However, any database writes that are pending at the end of the log, including updates from other units of recovery, are still written out during the forward phase of restart. Any updates that must be rolled-back, such as for an inflight or in-abort unit of recovery, are done during the backout phase of restart.

BACKOUT= Indicates whether to use the backward-log-recovery phase of DB2 restart, which rolls back any units of recovery that were in one of the following two states when DB2 was last stopped:
- Inflight (did not complete the first phase of commit)
- In-abort (had started but not finished an abort)

YES Allows backward-log recovery.

If you specify a cold start (by using the same value for STARTRBA and ENDRBA), no recovery processing is performed.

NO Terminates backward-log recovery before log records are processed.

CSRONLY Performs only the first and second phases of restart processing (log initialization and current-status rebuild). After these phases, the system status is displayed, and restart terminates. Some parts of the log initialization are not performed, including any updating of the log and display of STARTRBA and ENDRBA information.

When DB2 is restarted with this option in effect, the conditional restart control record is not deactivated. To prevent the control record from remaining active, use the DSNJU003 utility again with CRESTART CANCEL, or with CRESTART CREATE to create a new active control record.

NEWCAT Changes the VSAM catalog name in the BSDS.

VSAMCAT=*catalog-name*
Changes the VSAM catalog name entry in the BSDS.

catalog-name can be up to eight characters long. The first character must be alphabetic, and the remaining characters can be alphanumeric.

DDF Updates the LOCATION, LUNAME, and PASSWORD values in the BSDS. If you use this statement to insert new values into the BSDS, you must include at least the LOCATION and LUNAME in the DDF statement. To update an existing set of values, you need to include only those values that you want to change. The DDF record cannot be deleted from the BSDS after it has been added; it can only be modified.

NOPASSWD removes the DDF password from the DDF record in the BSDS. No other keywords can be used with NOPASSWD.

LOCATION=*location-name*
Changes the LOCATION value in the BSDS.

location-name specifies the name of your local DB2 site.

ALIAS=*alias-name :alias-port*
Specifies one or more alias names for the location. An alias name is

a name besides the location name that connect processing can accept. Specifying an alias name does not change the location identifier for a database object.

alias-name specifies from 1 to 8 alias names for the location name. *alias-name* cannot be one of the valid DSNJU003 keywords.

:alias-port specifies a TCP/IP port number for the alias that can be used by DDF to accept distributed requests. This value must be a decimal number between 1 and 65535, including 65535, and must be different than the values for the PORT and RESPORT options. Specify a value for *alias-port* when you want to identify a subset of data sharing members to which a distributed request can go. For more information about member-specific access, see *DB2 Data Sharing: Planning and Administration*.

You can add or replace aliases by respecifying the ALIAS option. The new list of names replaces the existing list.

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.

You cannot specify any other keyword with NOALIAS.

LUNAME=*luname*
Changes the LUNAME value in the BSDS.

luname specifies the LUNAME value. The LUNAME in the BSDS must always contain the value that identifies your local DB2 subsystem to the VTAM network.

PASSWORD= The DDF password follows VTAM convention, but DB2 restricts it to one to eight alphanumeric characters. The first character must be either a capital letter or an alphabetic extender. The remaining characters can consist of alphanumeric characters and alphabetic extenders.

password Optionally assigns a password to the distributed data facility communication record that establishes communications for a distributed data environment. See *VTAM for MVS/ESA Resource Definition Reference* for a description of the PRTCT=*password* option on the APPL definition statement that is used to define DB2 to VTAM.

NOPASSWD Removes the archive password protection for all archives that are created after this operation. It also removes a previously existing password from the DDF record. No other keyword can be used with NOPASSWD.

GENERIC=*gluname*
Replaces the value of the DB2 GENERIC LUNAME subsystem parameter in the BSDS.

gluname specifies the GENERIC LUNAME value.

NGENERIC Changes the DB2 GENERIC LUNAME to binary zeros in the BSDS, indicating that no VTAM generic LU name support is requested.

PORT Identifies the TCP/IP port number that is used by DDF to accept incoming connection requests. This value must be a decimal

number between 0 and 65535, including 65535; zero indicates that DDF's TCP/IP support is to be deactivated.

If DB2 is part of a data sharing group, all the members of the DB2 data sharing group must have the same value for PORT.

RESPORT Identifies the TCP/IP port number that is used by DDF to accept incoming DRDA two-phase commit resynchronization requests. This value must be a decimal number between 0 and 65535, including 65535; zero indicates that DDF's TCP/IP support is to be deactivated. If RESPORT is non-zero, RESPORT must not be the same as the value that is supplied on PORT.

For data sharing DB2 systems, RESPORT must be uniquely assigned to each DB2 member, so that no two DB2 members use the same TCP/IP port for two-phase commit resynchronization.

CHECKPT Allows updating of the checkpoint queue with the start checkpoint and end checkpoint log records.

Attention: This statement can override DB2's efforts to maintain data in a consistent state. Do not use the statement without understanding the conditional restart and checkpoint processing processes, which are described in Part 4 (Volume 1) of *DB2 Administration Guide*.

TIME=*time* **On the CHECKPT statement**, specifies the time that the start checkpoint record was written.

On the HIGHRBA statement, TIME specifies when the log record with the highest RBA was written to the log.

time specifies the time value. For timestamp format, see "STARTIME" on page 682.

HIGHRBA Updates the highest-written log RBA in either the active or archive log data sets.

Attention: This statement can override DB2's efforts to maintain data in a consistent state. Do not use the statement without understanding the conditional restart process, which is described in Part 4 (Volume 1) of *DB2 Administration Guide*.

OFFLRBA=*offlrba*

Specifies the highest-offloaded RBA in the archive log.

offlrba is a hexadecimal number of up to 12 characters. If you use fewer than 12 characters, leading zeros are added. The value must end with hexadecimal X'FFF'.

Before running DSNJU003

This section contains information you need to be aware of prior to running DSNJU003.

Environment

Execute the change log inventory utility only as a batch job when DB2 is not running. Changing a BSDS for a data-sharing member by using DSNJU003 might

cause a log read request from another data-sharing member to fail. The failure occurs only if the second member tries to access the changed BSDS before the first member is started.

Authorization required

The authorization ID of the DSNJU003 job must have the requisite RACF authorization.

Control statement

See "Syntax and options of the DSNJU003 control statement" on page 677 for DSNJU003 syntax and option descriptions.

Required and optional data sets

DSNJU003 recognizes DD statements with the following DD names:

JOBCAT or STEPCAT Specifies the catalog in which the bootstrap data sets (BSDSs) are cataloged. This statement is optional. Typically, the high-level qualifier of the BSDS name points to the ICF catalog that contains an entry for the BSDS.

SYSUT1 Specifies and allocates the bootstrap data set. This statement is required.

SYSUT2 Specifies and allocates a second copy of the bootstrap data set. This statement is required if you use dual BSDSs.

Dual BSDSs and DSNJU003: With each execution of DSNJU003, the BSDS timestamp field is updated with the current system time. If you run DSNJU003 separately for each copy of a dual copy BSDS, the timestamp fields are not synchronized, and DB2 fails at startup. If you change the contents of the BSDS copy by running DSNJU003, DB2 issues error message DSNJ122I. Therefore, if you use DSNJU003 to update dual copy BSDSs, update both BSDSs within a single execution of DSNJU003.

SYSPRINT Specifies a data set for print output. This statement is required. The logical record length (LRECL) is 125.

SYSIN Specifies the input data set for statements. This statement is required. The logical record length (LRECL) is 80.

Optional statements

The change log inventory utility provides the following statements:

- NEWLOG
- DELETE
- SYSTEMDB
- CRESTART
- NEWCAT
- DDF
- CHECKPT
- HIGHRBA

You can specify any statement one or more times. In each statement, separate the operation name from the first parameter by one or more blanks. You can use parameters in any order; separate them by commas with no blanks. Do not split a parameter description across two SYSIN records.

A statement that contains an asterisk in column 1 is considered a comment and is ignored. However, it appears in the output listing. To include a comment or sequence number in a SYSIN record, separate it from the last comma by a blank. When a blank is encountered after a comma, the rest of the record is ignored.

During execution of DSNJU003, a significant error in any statement causes that statement and all subsequent statements to be skipped. However, all remaining statements are checked for syntax errors. Therefore, BSDS updates are not made for any operation that is specified in the statement in error and in any subsequent statements.

Using DSNJU003 to modify the BSDS

This section describes the following tasks that are associated with running the DSNJU003 utility:

Running DSNJU003

Execute the utility with the following statement, which can be included only in a batch job:

```
//EXEC   PGM=DSNJU003
```

Making changes for active logs

Adding: If an active log is in stopped status, it is not reused for output logging; however, it continues to be used for reading. To add a new active log:

1. Use the Access Method Services DEFINE command to define new active log data sets.
2. Use DSNJLOGF to preformat the new active log data sets. **Restriction:** If you do not preformat these logs with the DSNJLOGF utility, DB2 needs to preformat them the first time they are used. Otherwise, performance might be impacted. This restriction applies to empty data sets and data sets with residual data.
3. Use DSNJU003 to register the new data sets in the BSDS.

For example, specify the following statements:

```
NEWLOG DSNAME=DSNC810.LOGCOPY1.DS04,COPY1
NEWLOG DSNAME=DSNC810.LOGCOPY2.DS04,COPY2
```

To copy the contents of an old active log data set to the new one, you can also give the RBA range and the starting and ending timestamp on the NEWLOG statement.

To archive to disk when the size of your active logs has increased, you might find it necessary to increase the size of your archive data set primary and secondary space quantities in DSNZPARM.

Deleting: To delete information about an active log data set from the BSDS, you might specify the following statements:

```
DELETE DSNAME=DSNC810.LOGCOPY1.DS01
DELETE DSNAME=DSNC810.LOGCOPY2.DS01
```

Recording: To record information about an existing active log data set in the BSDS, you might specify the following statement:

```
NEWLOG DSNAME=DSNC810.LOGCOPY2.DS05,COPY2,STARTIME=19910212205198,
             ENDTIME=19910412205200,STARTRBA=43F8000,ENDRBA=65F3000
```

You can insert a record of that information into the BSDS for any of these reasons:

- The data set has been deleted and is needed again.
- You are copying the contents of one active log data set to another data set (copy 1 to copy 2).
- You are recovering the BSDS from a backup copy.

Enlarging: When DB2 is inactive (down), use one of the following procedures.

If you can use the Access Method Services REPRO command, follow these steps:

1. Stop DB2. This step is required because DB2 allocates all active log data sets when it is active.
2. Use the Access Method Services ALTER command with the NEWNAME option to rename your active log data sets.
3. Use the Access Method Services DEFINE command to define larger active log data sets. Refer to installation job DSNTIJIN to see the definitions that create the original active log data sets. See *DB2 Installation Guide*.

   By reusing the old data set names, you don't need to run the change log inventory utility to establish new names in the BSDSs. The old data set names and the correct RBA ranges are already in the BSDSs.
4. Use the Access Method Services REPRO command to copy the old (renamed) data sets into their respective new data sets.
5. Start DB2.

If you cannot use the Access Method Services REPRO command, follow this procedure:

1. Ensure that all active log data sets except the current active log data sets have been archived. Active log data sets that have been archived are marked REUSABLE in print log map utility (DSNJU004) output.
2. Stop DB2.
3. Rename or delete the reusable active logs. Allocate new, larger active log data sets with the same names as the old active log data sets.
4. Run the DSNJLOGF utility to preformat the new log data sets.
5. Run the change log inventory utility (DSNJU003) with the DELETE statement to delete all active logs except the current active logs from the BSDS.
6. Run the change log inventory utility with the NEWLOG statement to add to the BSDS the active logs that you just deleted. So that the logs are added as empty, do not specify an RBA range.
7. Start DB2.
8. Issue the ARCHIVE LOG command to cause DB2 to truncate the current active logs and switch to one of the new sets of active logs.
9. Repeat steps 2 through 7 to enlarge the active logs that were just archived.

Although all log data sets do not need to be the same size, from an operational standpoint using the same size is more consistent and efficient. If the log data sets are not the same size, tracking your system's logs can be more difficult. Space can be wasted if you are using dual data sets of different sizes because they fill only to the size of the smallest, not using the remaining space on the larger one.

If you are archiving to disk and the size of your active logs has increased, you might need to increase the size of your archive log data sets. However, because of DFSMS disk management limits, you must specify less than 64 000 tracks for the primary space quantity. See the PRIMARY QUANTITY and SECONDARY QTY fields on installation panel DSNTIPA to modify the primary and secondary allocation space quantities. See *DB2 Installation Guide* for more information.

Making changes for archive logs

Adding: When the recovery of an object depends on reading an existing archive log data set, the BSDS must contain information about that data set, so that the recovery job can find it. To register information about an existing archive log data set in the BSDS, you might specify the following statement:

```
NEWLOG DSNAME=DSNC810.ARCHLOG1.D89021.T2205197.A0000015,COPY1VOL=DSNV04,
UNIT=TAPE,STARTRBA=3A190000,ENDRBA=3A1F0000,CATALOG=NO
```

Deleting: To delete an entire archive log data set from one or more volumes, you might specify the following statement:

```
DELETE DSNAME=DSNC810.ARCHLOG1.D89021.T2205197.A0000015,COPY1VOL=DSNV04
```

Creating a conditional restart control record

To create a new conditional restart control record in the BSDS, you must execute the change log inventory utility and use the CRESTART control statement. For example, to truncate the log, to specify the earliest log RBA, and to bypass backout, use a statement similar to the following statement:

```
CRESTART CREATE,STARTRBA=28894,ENDRBA=58000,BACKOUT=NO
```

To specify a cold start, make the values of STARTRBA and ENDRBA equal with a statement similar to the following statement:

```
CRESTART CREATE,STARTRBA=4A000,ENDRBA=4A000
```

In most cases when doing a cold start, you should make sure that the STARTRBA and ENDRBA are set to an RBA value that is greater than the highest used RBA.

An existing conditional restart control record governs any START DB2 operation until one of these events occurs:

- A restart operation completes.
- A CRESTART CANCEL statement is issued.
- A new conditional restart control record is created.

Deleting log data sets with errors

If an active log data set has encountered an I/O error, perform the following steps:

1. If you use dual active log data sets, check if the data from the bad active log data set is saved in the other active log. If it is, you can use the other active log.
2. If you cannot use the other active log or if the active log is in the STOPPED status, you must fix the problem manually.

a. Check to see if the data set has been offloaded. For example, check the list of archive log data sets to see if one has the same RBA range as the active log data set. This list can be created by using the DSNJU004 (print log map) utility.
b. If the data set has not been offloaded, copy the data to a new VSAM data set. If the data set has been offloaded, create a new VSAM data set that is to be used as an active log data set.
c. Specify DELETE to remove information about the bad data set from the BSDS.
d. Specify NEWLOG to identify the new data set as the new active log. The DELETE and NEWLOG operations can be performed by the same job step. (The DELETE statement precedes the NEWLOG statement in the SYSIN input data set.)

3. Delete the bad data set, using VSAM Access Method Services.

Use the print log map utility before and after running the change log inventory utility to ensure correct execution and to document changes.

When using dual active logs, choose a naming convention that distinguishes primary and secondary active log data set. The naming convention should also identify the log data sets within the series of primary or secondary active log data sets. For example, the default naming convention that is established at DB2 installation time is:

```
prefix.LOGCOPYn.DSmm
```

In this convention, *n*=1 for all primary log data sets, *n*=2 for all secondary log data sets, and *mm* is the data set number within each series.

If a naming convention such as the default convention is used, pairs of data sets with equal *mm* values are usually used together. For example, DSNC120.LOGCOPY1.DS02 and DSNC120.LOGCOPY2.DS02 are used together.

However, after you run the change log inventory utility with the DELETE and NEWLOG statements, the primary and secondary series can become unsynchronized, even if the NEWLOG data set name that you specify is the same as the old data set name. To avoid this situation, always do maintenance on both data sets of a pair in the same change log inventory execution:

- Delete both data sets together.
- Define both data sets together with NEWLOG statements.

The data sets themselves do not require deletion and redefinition.

To ensure consistent results, execute the change log inventory utility on the same z/OS system on which the DB2 online subsystem executes.

If misused, the change log inventory utility can compromise the viability and integrity of the DB2 subsystem. Only highly skilled people, such as the DB2 system administrator, should use this utility, and then only after careful consideration.

Before initiating a conditional restart or cold restart, you should consider making backup copies of all disk volumes that contain any DB2 data sets. This enables a possible fallback. The backup data sets must be generated when DB2 is not active.

Altering references to NEWLOG and DELETE data sets

Use the NEWLOG and DELETE statements to add and delete references to data sets in the BSDS. The log data sets are not changed in any way. If DELETE and NEWLOG are used for a reference in the BSDS to an active log data set, the referenced log data set itself does not require alteration.

Defining the high-level qualifier for catalog and directory objects

Use the NEWCAT statement to define the high-level qualifier that is to be used for the following objects:

- Catalog table spaces and index spaces
- Directory table spaces and index spaces

At startup, the DB2 system checks that the name that is recorded with NEWCAT in the BSDS is the high-level qualifier of the DB2 system table spaces that are defined in the load module for subsystem parameters.

NEWCAT is normally used only at installation time. See "Renaming DB2 system data sets" for an additional function of NEWCAT.

When you change the high-level qualifier by using the NEWCAT statement, you might specify the following statements:

```
//S2 EXEC PGM=DSNJU003
//SYSUT1   DD DSN=DSNC120.BSDS01,DISP=OLD
//SYSUT2   DD DSN=DSNC120.BSDS02,DISP=OLD
//SYSPRINT DD SYSOUT=*
  NEWCAT VSAMCAT=DBP1
```

After you run the change log inventory utility with the NEWCAT statement, the utility generates output similar to the following output:

```
NEWCAT VSAMCAT=DBP1
DSNJ210I OLD VASAM CATALOG NAME=DSNC120, NEW CATALOG NAME=DBP1
DSNJ225I NEWCAT OPERATION COMPLETED SUCCESSFULLY
DSNJ200I DSNJU003 CHANGE LOG INVENTORY UTILITY
         PROCESSING COMPLETED SUCCESSFULLY
```

Renaming DB2 system data sets

Occasionally, you might want to rename the DB2 system table spaces. In that case you should perform the following steps:

1. Stop DB2 in a consistent state.
2. Create a full system backup so that you can recover from operational errors.
3. Execute the change log inventory utility with NEWCAT.
4. Rename the BSDS and all DB2 directory and catalog table spaces and index spaces with IDCAMS.
5. Reassemble DSNZPARM to redefine the high-level qualifier for the system table spaces.
6. Update the BSDS name in the DB2 startup procedure.
7. Start DB2.
8. Drop and re-create the work file database.
9. Optionally use the ALTER command for table spaces in DSNDB04 and user databases.

Renaming DB2 active log data sets

When you rename system data sets, you might also want to rename the log data sets. In that case:

1. Stop DB2 in a consistent state.
2. Create a full system backup so that you can recover from operational errors.
3. Delete the reusable active log data sets with IDCAMS, but keep the current active log.
4. Define a new set of active log data sets with IDCAMS.
5. Execute the change log inventory utility to remove names of deleted active log data sets and to define the new active log data set names in the BSDS.
6. Start and use DB2 normally.

When the current active log is archived and becomes reusable, you can delete it.

Renaming DB2 archive log data sets

You do not need to rename archive log data sets for the following reasons:

- Old archive logs are replaced as a part of the normal maintenance cycle.
- The RECOVER utility works with archive logs that contain different high-level qualifiers.

To modify the high-level qualifier for archive log data sets, you need to reassemble DSNZPARM.

Sample DSNJU003 control statements

Example 1: Adding a new archive log data set. The following control statement specifies that the DSNJU003 utility is to add the data set DSNREPAL.A0001187 to the BSDS. The volume serial number for the data set is DSNV04, as indicated by the COPY1VOL option. The device type is SYSDA, and the data set is not to be cataloged. The RBA of the beginning of the archive log data set volume is 3A190000, and the end RBA is 3A1F0000.

```
//STEP5 EXEC PGM=DSNJU003,COND=EVEN
//SYSUT1   DD DSN=DSNCAT.BSDS01,DISP=SHR
//SYSUT2   DD DSN=DSNCAT.BSDS02,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
NEWLOG DSNAME=DSNREPAL.A0001187,COPY1VOL=DSNV04,UNIT=SYSDA,
STARTRBA=3A190000,ENDRBA=3A1F0000,CATALOG=NO
/*
```

Example 2: Deleting a data set. The following control statement specifies that DSNJU003 is to delete data set DSNREPAL.A0001187 from the BSDS. The volume serial number for the data set is DSNV04, as indicated by the COPY1VOL option.

```
DELETE DSNAME=DSNREPAL.A0001187,COPY1VOL=DSNV04
```

Example 3: Creating a new conditional restart control record. The following statement specifies that DSNJU003 is to create a new conditional restart control record, which controls the next restart of DB2. BACKOUT=NO indicates that DB2 is not to execute the backward-log-recovery phase when it restarts. The ENDRBA option indicates that 000000010000 is the last RBA of the log 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 this RBA is discarded.

```
CRESTART CREATE,BACKOUT=NO,ENDRBA=000000010000
```

Example 4: Adding a communication record to the BSDS. The following control statement specifies that DSNJU003 is to add a new communication record to the BSDS. The location, LU name, and password values are all provided.

```
DDF LOCATION=USIBMSTODB22,LUNAME=STL#M08,PASSWORD=$STL@290
```

Example 5: Adding a communication record with an alias to the BSDS. The following control statement specifies that DSNJU003 is to add a communication record to the BSDS. The location, alias, LU name, and password values are all provided.

```
DDF LOCATION=USIBMSTODB22,ALIAS=STL715A1,STL715A2,LUNAME=STL#M08,PASSWORD=$STL@290
```

Example 6: Adding multiple aliases and alias ports to the BSDS. The following control statement specifies five alias names for the communication record in the BSDS (MYALIAS1, MYALIAS2, MYALIAS3, MYALIAS4, and MYALIAS5). Only MYALIAS2 and MYALIAS5 support subsets of a data sharing group. Any alias names that were specified in a previous DSNJU003 utility job are removed.

```
DDF ALIAS=MYALIAS1,MYALIAS2:8002,MYALIAS3,MYALIAS4,MYALIAS5:10001
```

Example 7: Specifying a point in time for system recovery. The following control statement specifies that DSNJU003 is to create a new conditional restart control record. The SYSPITR option specifies an end RBA value as the point in time for system recovery for a non-data sharing system. For a data sharing system, use an end LRSN value instead of an end RBA value. This point in time is used by the RESTORE SYSTEM utility.

```
//JOBLIB  DD  DSN=USER.TESTLIB,DISP=SHR
//        DD  DSN=DSN810.SDSNLOAD,DISP=SHR
//STEP01 EXEC PGM=DSNJU003
//SYSUT1 DD DSN=DSNC810.BSDS01,DISP=OLD
//SYSUT2 DD DSN=DSNC810.BSDS02,DISP=OLD
//SYSPRINT DD SYSOUT=*
//SYSIN  DD *
  CRESTART CREATE,SYSPITR=04891665D000
/*
```

Example 8: Removing aliases from a communication record. The following control statement specifies that no alias names apply. Any alias names that were specified in a previous DSNJU003 utility job are removed.

```
DDF NOALIAS
```

Example 9: Updating the checkpoint queue. The following control statement specifies that DSNJU003 is to update the checkpoint queue with the start checkpoint and end checkpoint log records. The RBAs of these log records are indicated by the STARTRBA and ENDRBA options. The TIME option specifies the time that the start checkpoint record was written.

```
/*
//STEP3 EXEC PGM=DSNJU003
//SYSUT1   DD DSN=DSNCAT.BSDS01,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
    CHECKPT STARTRBA=0,ENDRBA=FFFFFFFFFFFF,
            TIME=19893652359599
/*
```

Only use the CHECKPT statement if you have a good understanding of conditional restart and checkpoint processing.

Chapter 37. DSNJU004 (print log map)

The print log map (DSNJU004) utility lists the following information:

- Log data set name, log RBA association, and log LRSN for both copy 1 and copy 2 of all active and archive log data sets
- Active log data sets that are available for new log data
- Status of all conditional restart control records in the bootstrap data set
- Contents of the queue of checkpoint records in the bootstrap data set
- The communication record of the BSDS, if one exists
- Contents of the quiesce history record
- System and utility timestamps
- Contents of the checkpoint queue
- Archive log command history
- BACKUP SYSTEM utility history
- System CCSID information

In a data sharing environment, the DSNJU004 utility can list information from any or all BSDSs of a data sharing group.

Additional information regarding the DSNJU004 utility appears in Part 4 (Volume 1) of *DB2 Administration Guide*.

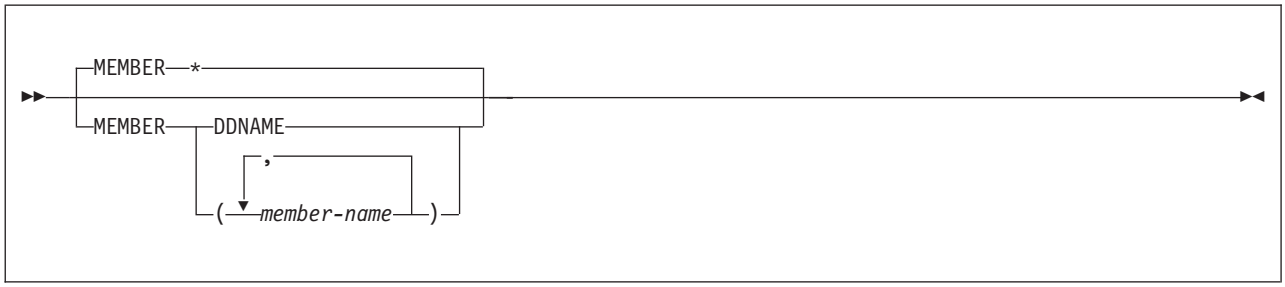
The following topics provide additional information:

- "Syntax and options of the DSNJU004 control statement"
- "Before running DSNJU004" on page 698
- "Sample DSNJU004 control statement" on page 699
- "DSNJU004 (print log map) output" on page 699

Syntax and options of the DSNJU004 control statement

Using the SYSIN data set allows you to list information from any or all BSDSs of a data sharing group.

DSNJU004 (print log map) syntax diagram



Option descriptions

The following keywords can be used in an optional control statement on the SYSIN data set:

MEMBER
Specifies which member's BSDS information to print.

*	Prints the information from the BSDS of each member in the data sharing group.
DDNAME	Prints information from only those BSDSs that are pointed to by the M*xx*BSDS DD statements.
(*member-name***)**	Prints information for only the named group members.

Before running DSNJU004

This section contains information that you need to be aware of prior to running DSNJU004.

Environment

The DSNJU004 program runs as a batch job.

This utility can be executed either when DB2 is running and when it is not running. However, to ensure consistent results from the utility job, the utility and the DB2 online subsystem must both be executing under the control of the same operating system.

Authorization required

The user ID of the DSNJU004 job must have requisite RACF authorization.

Control statement

See "DSNJU004 (print log map) syntax diagram" on page 697 for DSNJU004 syntax and option descriptions. See "Sample DSNJU004 control statement" on page 699 for an example of a control statement.

Required and optional data sets

DSNJU004 recognizes DD statements with the following DD names:

JOBCAT or STEPCAT Specifies the catalog in which the bootstrap data set (BSDS) is cataloged. This statement is optional. Typically, the high-level qualifier of the BSDS name points to the integrated catalog facility catalog that contains an entry for the BSDS.

SYSUT1 Specifies and allocates the bootstrap data set. This statement is required. It allocates the BSDS. If the BSDS must be shared with a concurrently executing DB2 online subsystem, use DISP=SHR on the DD statement.

SYSPRINT Specifies a data set or print spool class for print output. This statement is required. The logical record length (LRECL) is 125.

SYSIN (optional)
Contains the control statement. If you do not specify the SYSIN DD statement, BSDS information is printed only from the BSDS data set that is identified by the SYSUT1 DD statement.

GROUP Names a single BSDS. DB2 can use this BSDS to find the names of all BSDSs in the group. Ensure that the BSDS name that you specify is not the BSDS of a member that has been quiesced since before new members joined the group. This statement is required if the control statement specifies either of these options:
- MEMBER *
- MEMBER(member-name)

M*nn*BSDS Names the BSDS data set of a group member whose information is to be listed. You must specify one such DD statement for each member. The statements are required if the control statement specifies MEMBER DDNAME. *nn* represents a two-digit number. You must use consecutive two-digit numbers from 01 to the total number of required members. If a break occurs in the sequence of numbers, any number after the break is ignored.

Running the DSNJU004 utility

Use the following EXEC statement to execute this utility:

```
//   EXEC   PGM=DSNJU004
```

Recommendations

- For dual BSDSs, execute the print log map utility twice, once for each BSDS, to compare their contents.
- To ensure consistent results for this utility, execute the utility job on the same z/OS system on which the DB2 online subsystem executes.
- Execute the print log map utility regularly, possibly daily, to keep a record of recovery log data set usage.
- Use the print log map utility to document changes that are made by the change log inventory utility.

Sample DSNJU004 control statement

The following statement specifies that DSNJU004 is to print information from the BSDS for each member in the data sharing group:

```
//PLM      EXEC PGM=DSNJU004
//GROUP    DD DSN=DBD1.BSDS01,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
           MEMBER *
```

DSNJU004 (print log map) output

Figure 121 on page 700 and Figure 122 on page 702 show example output from the print log map utility. This output includes the following information:

- The data set name (DSN) of the BSDS.
- The system date and time (SYSTEM TIMESTAMP), which is set at the time that the subsystem stops.
- The date and time that the BSDS was last changed by the change log inventory utility (listed as the UTILITY TIMESTAMP).
- The integrated catalog facility catalog name that is associated with the BSDS.
- The highest-written RBA. The value is updated each time the log buffers are physically written to disk.
- The highest RBA that was offloaded.
- Log RBA ranges (STARTRBA and ENDRBA) and data set information for active and archive log data sets. The last active log data set shown is the current active log.
- Information about each active log data set. This information includes the starting and ending RBAs within the data set, the date and time the data set was created, and the data set's name (DSN), and status.

- Information about each archive log data set. This information includes the starting and ending RBAs within the data set, the date and time the data set was created, and the data set's name (DSN), unit and volume of storage, and status.
- Conditional restart control records. For a description of these records and the format of this part of the output from the print log map utility, see "Reading conditional restart control records" on page 707.
- The contents of the checkpoint description queue. For a description of this output, see Figure 125 on page 707.
- Archive log command history. For a description of this output, see Figure 124 on page 707.
- The distributed data facility (DDF) communication record. This record contains the DB2-defined location name, any alias names for the location name, and the VTAM-defined LU name. DB2 uses this information to establish the distributed database environment.
- The tokens for all BACKUP SYSTEM utility records. The token identifies each backup version that has been created.
- The RBA or LRSN when the subsystem was converted to enabling-new-function mode.

The sample print log map utility output in Figure 121 is for a non-data-sharing subsystem.

```
********************************************************************************
*                                                                              *
*   LOG MAP OF THE BSDS DATA SET BELONGING TO MEMBER 'NO NAME ' OF GROUP 'NO NAME '.   *
*                                                                              *
********************************************************************************
DSNJCNVB CONVERSION PROGRAM HAS NOT RUN   DDNAME=SYSUT1
   LOG MAP OF BSDS DATA SET COPY 1, DSN=DSNC810.BSDS01
   LTIME INDICATES LOCAL TIME, ALL OTHER TIMES ARE GMT.
         DATA SHARING MODE IS OFF
         SYSTEM TIMESTAMP   - DATE=2003.346  LTIME= 8:36:35.35
         UTILITY TIMESTAMP  - DATE=2003.346  LTIME= 8:18:10.10
         VSAM CATALOG NAME=DSNC810
         HIGHEST RBA WRITTEN       000004FD1B40  2003.346  16:36:26.1
         HIGHEST RBA OFFLOADED     0000031B1FFF
         RBA WHEN CONVERTED TO V4  000000000000
       THIS BSDS HAS MEMBER RECORDS FOR THE FOLLOWING MEMBERS:
         HOST MEMBER NAME:
           MEMBER ID:                0
           GROUP NAME:
           BSDS COPY 1 DATA SET NAME:
           BSDS COPY 2 DATA SET NAME:
           ENFM START RBA/LRSN:      0000035B8D5C
ACTIVE LOG COPY 1 DATA SETS
     START RBA/TIME         END RBA/TIME        DATE    LTIME DATA SET INFORMATION
  -------------------  -------------------  --------  ----- -------------------
    0000048C4000           000004C47FFF       2001.045  14:39 DSN=DSNC810.LOGCOPY1.DS02
  2003.346  16:33:11.8  2003.346  16:34:09.4  PASSWORD=(NULL) STATUS=REUSABLE
    000004C48000           000004FC8FFF       2001.045  14:39 DSN=DSNC810.LOGCOPY1.DS03
  2003.346  16:34:09.4  2003.346  16:35:47.8  PASSWORD=(NULL) STATUS=TRUNCATED, REUSABLE
    000004FC9000           00000534CFFF       2001.045  14:39 DSN=DSNC810.LOGCOPY1.DS01
  2003.346  16:35:47.8  ........  ..........  PASSWORD=(NULL) STATUS=REUSABLE
```

Figure 121. Sample print log map utility output for a non-dating-sharing subsystem (Part 1 of 3)

```
ARCHIVE LOG COPY 1 DATA SETS
    START RBA/TIME         END RBA/TIME        DATE    LTIME DATA SET INFORMATION
  -------------------  -------------------  --------  ----- --------------------
    000002A30000           000002DB3FFF       2003.346   8:20 DSN=DSNC810.ARCHLOG1.A0000001
  2001.178  15:46:36.8  2001.178  15:51:03.9                  PASSWORD=(NULL) VOL=SCR03 UNIT=SYSDA
                                                             CATALOGUED
    000002DB4000           000003137FFF       2003.346   8:20 DSN=DSNC810.ARCHLOG1.A0000002
  2001.178  15:51:03.9  2003.346  16:18:43.8                  PASSWORD=(NULL) VOL=SCR03 UNIT=SYSDA
                                                             CATALOGUED
    000003138000           000003167FFF       2003.346   8:20 DSN=DSNC810.ARCHLOG1.A0000003
  2003.346  16:18:43.8  2003.346  16:20:49.5                  PASSWORD=(NULL) VOL=SCR03 UNIT=SYSDA
                                                             CATALOGUED
    000003168000           00000316BFFF       2003.346   8:20 DSN=DSNC810.ARCHLOG1.A0000004
  2003.346  16:20:49.5  2003.346  16:20:49.6                  PASSWORD=(NULL) VOL=SCR03 UNIT=SYSDA
                                                             CATALOGUED
    00000316C000           0000031ABFFF       2003.346   8:21 DSN=DSNC810.ARCHLOG1.A0000005
  2003.346  16:20:49.6  2003.346  16:21:23.7                  PASSWORD=(NULL) VOL=SCR03 UNIT=SYSDA
                                                             CATALOGUED
    0000031AC000           0000031B1FFF       2003.346   8:21 DSN=DSNC810.ARCHLOG1.A0000006
  2003.346  16:21:23.7  2003.346  16:21:23.8                  PASSWORD=(NULL) VOL=SCR03 UNIT=SYSDA
                                                             CATALOGUED
ACTIVE LOG COPY 2 DATA SETS
    START RBA/TIME         END RBA/TIME        DATE    LTIME DATA SET INFORMATION
  -------------------  -------------------  --------  ----- --------------------
    0000048C4000           000004C47FFF       2001.045  14:39 DSN=DSNC810.LOGCOPY2.DS02
  2003.346  16:33:11.8  2003.346  16:34:09.4  PASSWORD=(NULL) STATUS=REUSABLE
    000004C48000           000004FC8FFF       2001.045  14:39 DSN=DSNC810.LOGCOPY2.DS03
  2003.346  16:34:09.4  2003.346  16:35:47.8  PASSWORD=(NULL) STATUS=TRUNCATED, REUSABLE
    000004FC9000           00000534CFFF       2001.045  14:39 DSN=DSNC810.LOGCOPY2.DS01
  2003.346  16:35:47.8  ........  ..........  PASSWORD=(NULL) STATUS=REUSABLE
ARCHIVE LOG COPY 2 DATA SETS
NO ARCHIVE DATA SETS DEFINED FOR THIS COPY
DSNJ401I  DSNRJPCR RESTART CONTROL RECORD NOT FOUND
                         CHECKPOINT QUEUE
                      16:36:52 DECEMBER 12, 2003
       TIME OF CHECKPOINT       16:36:15 DECEMBER 12, 2003
       BEGIN CHECKPOINT RBA            000004FCE074
       END CHECKPOINT RBA              000004FD1B40
       SHUTDOWN CHECKPOINT
       TIME OF CHECKPOINT       16:35:47 DECEMBER 12, 2003
       BEGIN CHECKPOINT RBA            000004FCAA17
       END CHECKPOINT RBA              000004FCD90C
       TIME OF CHECKPOINT       16:35:13 DECEMBER 12, 2003
       BEGIN CHECKPOINT RBA            000004F4EABB
       END CHECKPOINT RBA              000004F8B994
       ...
       TIME OF CHECKPOINT       15:50:56 JUNE 27, 2001
       BEGIN CHECKPOINT RBA            000002D6D9F1
       END CHECKPOINT RBA              000002D72066
       TIME OF CHECKPOINT       15:50:53 JUNE 27, 2001
       BEGIN CHECKPOINT RBA            000002D4861E
       END CHECKPOINT RBA              000002D4C6D4
```

Figure 121. Sample print log map utility output for a non-dating-sharing subsystem (Part 2 of 3)

```
                     ARCHIVE LOG COMMAND HISTORY
                      16:36:54 DECEMBER 12, 2003
   DATE          TIME          RBA         MODE   WAIT   TIME
------------  ----------  ------------   -------  ----   -----
DEC 12, 2003  16:35:47.8  000004FC89E5   QUIESCE  YES    999
DEC 12, 2003  16:31:49.5  00000453F379
DEC 12, 2003  16:21:23.8  0000031B1388
DEC 12, 2003  16:21:23.7  0000031AB392
DEC 12, 2003  16:20:49.6  00000316B000
DEC 12, 2003  16:20:49.5  000003167000
                 **** DISTRIBUTED DATA FACILITY ****
                        COMMUNICATION RECORD
                      16:36:54 DECEMBER 12, 2003
LOCATION=STLEC1 ALIAS=(NULL)
LUNAME=SYEC1DB2 PASSWORD=DB2PW1 GENERICLU=(NULL) PORT=NULL RPORT=NULL
DSNJ401I  DSNUPBHR BACKUP SYSTEM UTILITY HISTORY RECORD NOT FOUND
                          SYSTEM CCSIDS
                      16:36:54 DECEMBER 12, 2003
    SYSTEM CCSIDS
    --------------------
    ASCII SBCS   = 1252
    ASCII MIXED  = 65534
    ASCII DBCS   = 65534
    EBCDIC SBCS  = 37
    EBCDIC MBCS  = 65534
    EBCDIC DBCS  = 65534
    UNICODE SBCS = 367
    UNICODE MBCS = 1208
    UNICODE DBCS = 1200
DSNJ200I  DSNJU004 PRINT LOG UTILITY PROCESSING COMPLETED SUCCESSFULLY
```

Figure 121. Sample print log map utility output for a non-dating-sharing subsystem (Part 3 of 3)

The sample print log map utility output in Figure 122 is for a member of a data sharing group.

```
*********************************************************************************************
*                                                                                           *
*   LOG MAP OF THE BSDS DATA SET BELONGING TO MEMBER 'V81A    ' OF GROUP 'DSNCAT  '.    *
*                                                                                           *
*********************************************************************************************
DSNJCNVB CONVERSION PROGRAM HAS NOT RUN   DDNAME=SYSUT1
   LOG MAP OF BSDS DATA SET COPY 1, DSN=DSNC810.BSDS01
   LTIME INDICATES LOCAL TIME, ALL OTHER TIMES ARE GMT.
         DATA SHARING MODE IS ON
         SYSTEM TIMESTAMP   - DATE=2003.346  LTIME=10:45:53.34
         UTILITY TIMESTAMP  - DATE=2003.346  LTIME= 8:52:00.63
         VSAM CATALOG NAME=DSNC810
         HIGHEST RBA WRITTEN       0000068F92DE  2003.346  18:45:22.8
         HIGHEST RBA OFFLOADED     000000000000
         RBA WHEN CONVERTED TO V4  000001251868
         MAX RBA FOR TORBA         000001251868
         MIN RBA FOR TORBA         000000000000
         STCK TO LRSN DELTA        000000000000
```

Figure 122. Sample print log map utility output for a member of a data sharing group (Part 1 of 3)

```
THIS BSDS HAS MEMBER RECORDS FOR THE FOLLOWING MEMBERS:
          HOST MEMBER NAME:            V81A
            MEMBER ID:                 1
            GROUP NAME:                DSNCAT
            BSDS COPY 1 DATA SET NAME: DSNC810.BSDS01
            BSDS COPY 2 DATA SET NAME: DSNC810.BSDS02
            ENFM START RBA/LRSN:       BA75C10BCA2B
          MEMBER NAME:                 V81B
            MEMBER ID:                 2
            GROUP NAME:                DSNCAT
            BSDS COPY 1 DATA SET NAME: DSNC818.BSDS01
            BSDS COPY 2 DATA SET NAME: DSNC818.BSDS02
ACTIVE LOG COPY 1 DATA SETS
  START RBA/LRSN/TIME   END RBA/LRSN/TIME      DATE    LTIME DATA SET INFORMATION
 -------------------  -------------------  --------  ----- --------------------
   000005EB6000           000006239FFF       2001.045  14:39 DSN=DSNC810.LOGCOPY1.DS03
     BA75CC4E3821           BA75CF112B6E     PASSWORD=(NULL) STATUS=REUSABLE
 2003.346  18:13:45.8  2003.346  18:26:07.1
   00000623A000           0000065BDFFF       2001.045  14:39 DSN=DSNC810.LOGCOPY1.DS01
     BA75CF112B6F           BA75D0BD5A8A     PASSWORD=(NULL) STATUS=REUSABLE
 2003.346  18:26:07.1  2003.346  18:33:36.1
   0000065BE000           000006941FFF       2001.045  14:39 DSN=DSNC810.LOGCOPY1.DS02
     BA75D0BD5A8B           ............     PASSWORD=(NULL) STATUS=REUSABLE
 2003.346  18:33:36.1  ........  ..........
ARCHIVE LOG COPY 1 DATA SETS
NO ARCHIVE DATA SETS DEFINED FOR THIS COPY
ACTIVE LOG COPY 2 DATA SETS
  START RBA/LRSN/TIME   END RBA/LRSN/TIME      DATE    LTIME DATA SET INFORMATION
 -------------------  -------------------  --------  ----- --------------------
   000005EB6000           000006239FFF       2001.045  14:39 DSN=DSNC810.LOGCOPY2.DS03
     BA75CC4E3821           BA75CF112B6E     PASSWORD=(NULL) STATUS=REUSABLE
 2003.346  18:13:45.8  2003.346  18:26:07.1
   00000623A000           0000065BDFFF       2001.045  14:39 DSN=DSNC810.LOGCOPY2.DS01
     BA75CF112B6F           BA75D0BD5A8A     PASSWORD=(NULL) STATUS=REUSABLE
 2003.346  18:26:07.1  2003.346  18:33:36.1
   0000065BE000           000006941FFF       2001.045  14:39 DSN=DSNC810.LOGCOPY2.DS02
     BA75D0BD5A8B           ............     PASSWORD=(NULL) STATUS=REUSABLE
 2003.346  18:33:36.1  ........  ..........
ARCHIVE LOG COPY 2 DATA SETS
NO ARCHIVE DATA SETS DEFINED FOR THIS COPY
DSNJ401I  DSNRJPCR RESTART CONTROL RECORD NOT FOUND
                         CHECKPOINT QUEUE
                     18:46:25 DECEMBER 12, 2003
       TIME OF CHECKPOINT       18:45:08 DECEMBER 12, 2003
       BEGIN CHECKPOINT RBA             0000068F56FA
       END CHECKPOINT RBA               0000068F92DE
       END CHECKPOINT LRSN              BA75D35F5638
       SHUTDOWN CHECKPOINT
       TIME OF CHECKPOINT       18:41:20 DECEMBER 12, 2003
       BEGIN CHECKPOINT RBA             0000068EE29E
       END CHECKPOINT RBA               0000068F124A
       END CHECKPOINT LRSN              BA75D278B6BD
       ...
       TIME OF CHECKPOINT       01:00:21 JUNE 28, 2001
       BEGIN CHECKPOINT RBA             0000047319AF
       END CHECKPOINT RBA               000004736215
       END CHECKPOINT LRSN              B60D1C57A2EC
       TIME OF CHECKPOINT       01:00:17 JUNE 28, 2001
       BEGIN CHECKPOINT RBA             00000470E2B0
       END CHECKPOINT RBA               0000047128FC
       END CHECKPOINT LRSN              B60D1C545549
```

Figure 122. Sample print log map utility output for a member of a data sharing group (Part 2 of 3)

```
| ARCHIVE LOG COMMAND HISTORY
|                            MEMBER  V81A
|             DATA SHARING GROUP DSNCAT   CONTAINS 2 MEMBERS
|                         18:46:29 DECEMBER 12, 2003
|  DATE/SDATE   TIME/STIME      RBA        MODE    WAIT  TIME   SCOPE   CMD ORIGIN   STATUS        ACTIVE
| ------------  ----------  ------------  -------  ----  -----  -----   ----------   ------        ------
| DEC 12, 2003  17:24:35.5  0000050A56FA  QUIESCE  YES   999      G     V81A         ORIGINATOR      2
| DEC 12, 2003  17:01:29.5  000004ABD642                           M     V81A         ORIGINATOR
|                    **** DISTRIBUTED DATA FACILITY ****
|                           COMMUNICATION RECORD
|                         18:46:29 DECEMBER 12, 2003
| LOCATION=STLEC1 ALIAS=(NULL)
| LUNAME=SYEC1DB2 PASSWORD=DB2PW1 GENERICLU=SYEC1GLU PORT=NULL RPORT=NULL
| DSNJ401I  DSNUPBHR BACKUP SYSTEM UTILITY HISTORY RECORD NOT FOUND
|                             SYSTEM CCSIDS
|                         18:46:30 DECEMBER 12, 2003
|     SYSTEM CCSIDS
|     --------------------
|     ASCII SBCS   = 1252
|     ASCII MIXED  = 65534
|     ASCII DBCS   = 65534
|     EBCDIC SBCS  = 37
|     EBCDIC MBCS  = 65534
|     EBCDIC DBCS  = 65534
|     UNICODE SBCS = 367
|     UNICODE MBCS = 1208
|     UNICODE DBCS = 1200
| DSNJ200I  DSNJU004 PRINT LOG UTILITY PROCESSING COMPLETED SUCCESSFULLY
```

Figure 122. Sample print log map utility output for a member of a data sharing group (Part 3 of 3)

Timestamps in the BSDS

The output of the print log map utility reveals that many timestamps are recorded in the BSDS. Those timestamps record the date and time of various system events.

Timestamps in the output column LTIME are in local time. All other timestamps are in Greenwich Mean Time (GMT).

Figure 121 on page 700 and Figure 122 on page 702 show example output from the print log map utility. The following timestamps are included in the header section of the reports:

System timestamp

Reflects the date and time that the BSDS was last updated. The BSDS can be updated by several events:

- DB2 startup.
- During log write activities, whenever the write threshold is reached.

  Depending on the number of output buffers that you have specified and the system activity rate, the BSDS might be updated several times a second, or it might not be updated for several seconds, minutes, or even hours.
- Due to an error, DB2 might drop into single-BSDS mode from its normal dual BSDS mode. This action might occur when a request to get, insert, point to, update, or delete a BSDS

record is unsuccessful. When this error occurs, DB2 updates the timestamp in the remaining BSDS to force a timestamp mismatch with the disabled BSDS.

Utility timestamp The date and time that the contents of the BSDS were altered by the change log inventory utility (DSNJU003).

The following timestamps are included in the active and archive log data sets portion of the reports:

Active log date The date on which the active log data set was originally allocated on the DB2 subsystem.

Active log time The time at which the active log data set was originally allocated on the DB2 subsystem.

Archive log date The date of creation (not allocation) of the archive log data set.

Archive log time The time of creation (not allocation) of the archive log data set.

The following timestamps are included in the conditional restart control record portion of the report that is shown in Figure 126 on page 708:

Conditional restart control record
The current time and date. This data is reported for information only and is not kept in the BSDS.

CRCR created The time and date of creation of the CRCR by the CRESTART option in the change log inventory utility.

Begin restart The time and date that the conditional restart was attempted.

End restart The time and date that the conditional restart ended.

STARTRBA (timestamp) The time at which the control interval was written.

ENDRBA (timestamp) The time at which the last control interval was written.

Time of checkpoint The time and date that are associated with the checkpoint record that was used during the conditional restart process.

The following timestamps are included in the checkpoint queue and the DDF communication record sections of the report that is shown in Figure 125 on page 707:

Checkpoint queue The current time and date. This data is reported for information only and is not kept in the BSDS.

Time of checkpoint The time and date that the checkpoint was taken.

DDF communication record (heading)
The current time and date. This data is reported for information only, and is not kept in the BSDS.

Active log data set status

The BSDS records the status of an active log data set as one of the status values that are listed in Table 139. This table lists each status value and its meaning.

Table 139. Statuses of active log data sets

Status	Meaning
NEW	The data set has been defined but never used by DB2, or the log is truncated at a point before the data set was created. In either case, the data set starting and ending RBA values are reset to zero.
REUSABLE	Either the data set is new and has no records, or the data set has been offloaded. In the print log map output, the start RBA value for the last REUSABLE data set is equal to the start RBA value of the last archive log data set.
NOT REUSABLE	The data set contains records that have not been offloaded.
STOPPED	The offload processor encountered an error while reading a record, and that record could not be obtained from the other copy of the active log. Alternatively, an error occurred during truncation of the data set following a write I/O error. See Part 4 (Volume 1) of *DB2 Administration Guide*.
TRUNCATED	One of these conditions exists: • An I/O error occurred, and DB2 has stopped writing to this data set. The active log data set is offloaded, beginning with the starting RBA and continuing up to the last valid record segment in the truncated active log data set. (The RBA of the last valid record segment is less than the ending RBA of the active log data set.) Logging is switched to the next available active log data set and continues uninterrupted. • The log was truncated by a conditional restart at a point within the data set RBA range. • The DB2 ARCHIVE LOG command was issued while this data set was the current active log data set.

The status value for each active log data set is displayed in the print log map utility output. The sample print log map output in Figure 123 shows how the status is displayed.

```
ACTIVE LOG COPY 1 DATA SETS
    START RBA/TIME         END RBA/TIME        DATE    LTIME DATA SET INFORMATION
 --------------------  --------------------  --------  ----- --------------------
   00000316C000           0000031ABFFF       2001.045  14:39 DSN=DSNC810.LOGCOPY1.DS02
 2003.346  16:20:49.6  2003.346  16:21:23.7  PASSWORD=(NULL) STATUS=TRUNCATED, REUSABLE
   0000031AC000           0000031B1FFF       2001.045  14:39 DSN=DSNC810.LOGCOPY1.DS03
 2003.346  16:21:23.7  2003.346  16:21:23.8  PASSWORD=(NULL) STATUS=TRUNCATED, REUSABLE
   0000031B2000           000003535FFF       2001.045  14:39 DSN=DSNC810.LOGCOPY1.DS01
 2003.346  16:21:23.8  ........  ..........  PASSWORD=(NULL) STATUS=NOTREUSABLE
```

Figure 123. Portion of print log map utility output that shows active log data set status

Archive log command history

The print log map utility output also displays the archive log command history, as shown in Figure 124 on page 707.

```
ARCHIVE LOG COMMAND HISTORY
                      16:36:54 DECEMBER 12, 2003
   DATE          TIME          RBA         MODE   WAIT   TIME
------------  ----------  ------------   -------  ----   -----
DEC 12, 2003  16:35:47.8  000004FC89E5   QUIESCE  YES    999
DEC 12, 2003  16:31:49.5  00000453F379
DEC 12, 2003  16:21:23.8  0000031B1388
DEC 12, 2003  16:21:23.7  0000031AB392
DEC 12, 2003  16:20:49.6  00000316B000
DEC 12, 2003  16:20:49.5  000003167000
```

Figure 124. Portion of print log map utility output that shows archive log command history

The values in the TIME column of the ARCHIVE LOG COMMAND HISTORY section of the report in Figure 124 represent the time that the ARCHIVE LOG command was issued. This time value is saved in the BSDS and is converted to printable format at the time that the print log map utility is run. Therefore this value, when printed, can differ from other time values that were recorded concurrently. Some time values are converted to printable format when they are recorded, and then they are saved in the BSDS. These printed values remain the same when the printed report is run.

Reading conditional restart control records

In addition to listing information about log records, the print log map utility lists information about each conditional restart control record and each checkpoint. A sample description of a checkpoint record in the queue is shown in Figure 125.

```
                    CHECKPOINT QUEUE
                  15:54:57 FEBRUARY 04, 2003
TIME OF CHECKPOINT       15:54:37 FEBRUARY 04, 2003
BEGIN CHECKPOINT RBA            0000400000EC
END CHECKPOINT RBA              00004000229A
TIME OF CHECKPOINT       15:53:44 FEBRUARY 04, 2003
BEGIN CHECKPOINT RBA            00000B39E1EC
END CHECKPOINT RBA              00000B3A80A6
SHUTDOWN CHECKPOINT
TIME OF CHECKPOINT       15:49:40 FEBRUARY 04, 2003
BEGIN CHECKPOINT RBA            00000B2E33E5
END CHECKPOINT RBA              00000B2E9C88
...
TIME OF CHECKPOINT       21:06:01 FEBRUARY 03, 2003
BEGIN CHECKPOINT RBA            00000A7AA19C
END CHECKPOINT RBA              00000A82C998
```

Figure 125. Sample print log map description of checkpoints

A sample description of a conditional restart control record is shown in Figure 126 on page 708.

```
CRCR IDENTIFIER  0001
    USE COUNT   1
    RECORD STATUS
        CRCR NOT ACTIVE
        SUCCESSFUL RESTART
    PROCESSING STATUS
        COLD START (STARTRBA = ENDRBA)
        FORWARD =  NO
        BACKOUT =  NO
    STARTRBA                          000040000000
    ENDRBA                            000040000000
    ENDLRSN                           NOT SPECIFIED
    EARLIEST REQUESTED RBA            000000000000
    FIRST LOG RECORD RBA              000000000000
    ORIGINAL CHECKPOINT RBA           000000000000
    NEW CHECKPOINT RBA (CHKPTRBA)     NOT SPECIFIED
    CRCR CREATED              15:54:13 FEBRUARY 04, 2003
    BEGIN RESTART             15:54:26 FEBRUARY 04, 2003
    END RESTART               15:54:37 FEBRUARY 04, 2003
    RESTART PROGRESS                   STARTED     ENDED
                                       =======     =====
        CURRENT STATUS REBUILD           YES         YES
        FORWARD RECOVERY PHASE           YES         YES
        BACKOUT RECOVERY PHASE           YES         YES
```

Figure 126. Sample print log map description of a CRCR

Chapter 38. DSN1CHKR

The DSN1CHKR utility verifies the integrity of DB2 directory and catalog table spaces. DSN1CHKR scans the specified table space for broken links, broken hash chains, and records that are not part of any link or chain.

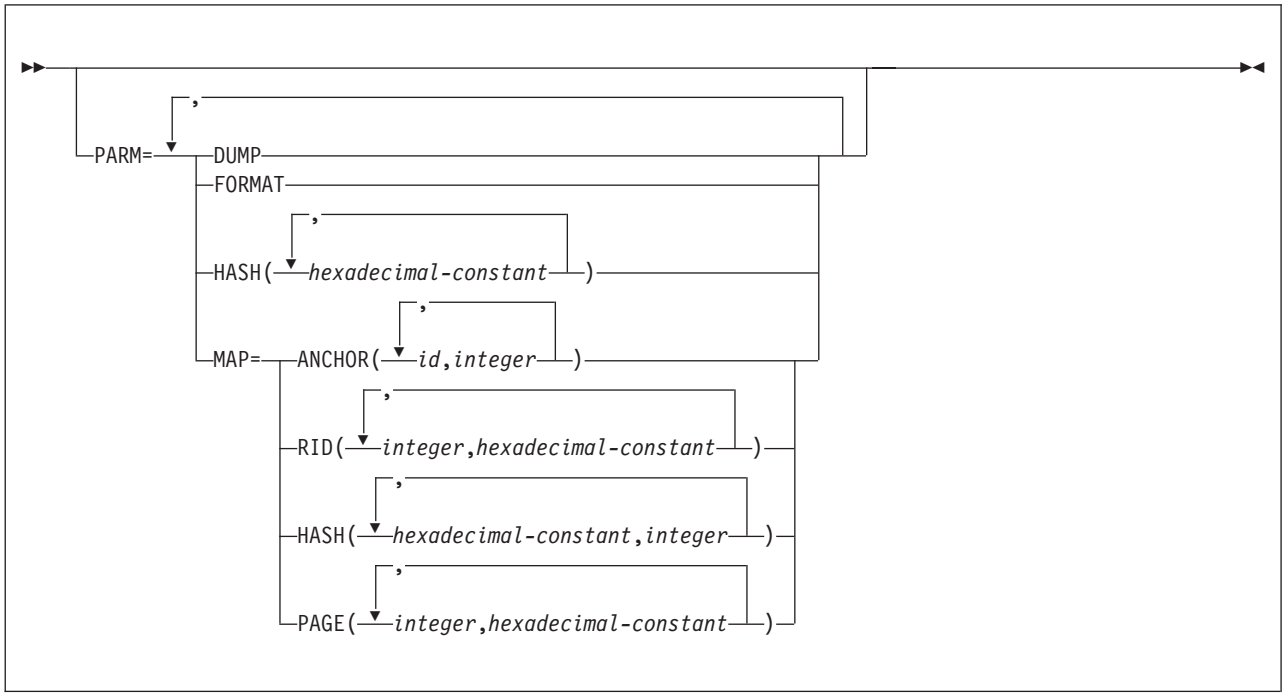
Use DSN1CHKR on a regular basis to promptly detect any damage to the catalog and directory.

The following topics provide additional information:

- "Syntax and options of the DSN1CHKR control statement"
- "Before running DSN1CHKR" on page 711
- "Sample DSN1CHKR control statements" on page 712
- "DSN1CHKR output" on page 715

Syntax and options of the DSN1CHKR control statement

DSN1CHKR syntax diagram



Option descriptions

The following parameters are optional. Specify parameters on the EXEC statement in any order after the required JCL parameter PARM=. If you specify more than one parameter, separate them with commas but no blanks. If you do not specify any parameters, DSN1CHKR scans all table space pages for broken links and for records that are not part of any link or chain, and prints the appropriate diagnostic messages.

DUMP Specifies that printed table space pages, if any, are to be in dump format. If you specify DUMP, you cannot specify the FORMAT parameter.

FORMAT Specifies that printed table space pages, if any, are to be formatted on output. If you specify FORMAT, you cannot specify the DUMP parameter.

HASH(*hexadecimal-constant***, ...)**
Specifies a hash value for a hexadecimal database identifier (DBID) in table space DBD01. DSN1CHKR returns hash values for each DBID in page form and in anchor point offset form.

hexadecimal-constant is the hash value for a DBID. The maximum number of DBIDs is 10.

MAP= Identifies a record whose pointer is to be followed. DSN1CHKR prints each record as it follows the pointer. Use this parameter only after you have determined which chain is broken. You can determine if a chain is broken by running DSN1CHKR without any parameters, or with only FORMAT or DUMP.

The options for this parameter help DSN1CHKR locate the record whose pointer it follows. Each option must point to the beginning of the 6-byte prefix area of a valid record or to the beginning of the hash anchor. If the value that you specify does not point to one of these, DSN1CHKR issues an error message and continues with the next pair of values.

ANCHOR(*id,integer***)**
Specifies the anchor point that DSN1CHKR is to map.

id identifies the starting page and anchor point in the form *ppppppaa*, where *pppppp* is the page number, and *aa* is the anchor point number.

integer determines which pointer to follow while mapping. 0 specifies the forward pointer; 4 specifies the backward pointer.

The maximum number of pairs is five.

RID(*integer, hexadecimal-constant***, ...)**
Identifies the record or hash anchor from which DSN1CHKR is to start mapping.

integer is the page and record, in the form *pppppprr*, where *pppppp* is the page number, and *rr* is the record number. These values are in hexadecimal format.

hexadecimal-constant specifies the hexadecimal displacement from the beginning of the record to the pointer in the record from which mapping starts.

The maximum number of pairs is five.

HASH(*hexadecimal-constant, integer***, ...)**
Specifies the value that DSN1CHKR is to hash and map for table space DBD01.

hexadecimal constant is the database identifier in table space DBD01.

integer determines which pointer to follow while mapping. 0 specifies the forward pointer; 4 specifies the backward pointer.

The maximum number of pairs is five.

PAGE(*integer, hexadecimal-constant***, ...)**
integer specifies the page number on which the record or hash anchor is to be located.

hexadecimal-constant specifies the offset to the pointer from the beginning of the page.

When you use the PAGE option, DSN1CHKR follows the forward pointer while mapping. If a forward pointer does not exist, DSN1CHKR stops mapping after the first record.

The maximum number of pairs is five.

Before running DSN1CHKR

The information in this section is necessary for running DSN1CHKR.

DSN1CHKR is a diagnosis tool; it executes outside the control of DB2. You should have detailed knowledge of DB2 data structures to make proper use of this service aid.

Environment

Run the DSN1CHKR program as a z/OS job.

Do not run DSN1CHKR on a table space while it is active under DB2. While DSN1CHKR runs, do not run other database operations for the database and table space that are to be checked. Use the STOP DATABASE command for the database and table space that are to be checked.

Authorization required

This utility does not require authorization. However, if RACF protects any of the data sets, the authorization ID must also have the necessary RACF authority.

Control statement

Create the utility control statement for the DSN1CHKR job. See "Syntax and options of the DSN1CHKR control statement" on page 709 for DSN1CHKR syntax and option descriptions.

Required data sets: DSN1CHKR uses two data definition (DD) statements. Specify the data set for the utility's output with the SYSPRINT DD statement. Specify the first data set piece of the table space that is to be checked with the SYSUT1 DD statement.

SYSPRINT Defines the data set that contains output messages from the DSN1CHKR program and all hexadecimal dump output.

SYSUT1 Defines the input data set. This data set can be a DB2 data set or a copy that is created by the DSN1COPY utility. Specify disposition of this data set as DISP=OLD to ensure that it is not in use by DB2. Set the data set's disposition as DISP=SHR only when the STOP DATABASE command has stopped the table space you want to check.

Restrictions

This section contains restrictions that you should be aware of before running DSN1CHKR.

Running DSN1COPY before DSN1CHKR

DSN1CHKR requires a VSAM data set as input; it cannot check a physical sequential data set.

DSN1CHKR does not use full image copies that are created with the COPY utility. If you create a full image copy with SHRLEVEL REFERENCE, you can copy it into a VSAM data set with DSN1COPY and check it with DSN1CHKR.

DSN1CHKR cannot use full image copies that are created with DFSMSdss concurrent copy. The DFSMSdss data set does not copy to a VSAM data set because of incompatible formats.

Recommendation: First copy the stopped table space to a temporary data set by using DSN1COPY. Use the DB2 naming convention for the copied data set. Run DSN1CHKR on the copy, which frees the actual table space for restart to DB2.

When you run DSN1COPY, use the CHECK option to examine the table space for page integrity errors. Although DSN1CHKR does check for these errors, running DSN1COPY with CHECK prevents an unnecessary invocation of DSN1CHKR.

Running DSN1CHKR on a valid table space

Run DSN1CHKR only on the following valid table spaces:

- DSNDB01.DBD01
- DSNDB06.SYSDBASE
- DSNDB06.SYSDBAUT
- DSNDB06.SYSGROUP
- DSNDB06.SYSPLAN
- DSNDB06.SYSVIEWS

Sample DSN1CHKR control statements

Example 1: Running DSN1CHKR on a temporary data set. In the sample JCL in Figure 127 on page 713, STEP1 allocates a temporary data set. The fifth qualifier in the data set name can be either I0001 or J0001. This example uses I0001. STEP2 stops database DSNDB06 with the STOP DATABASE command. STEP3 copies the target table space into the temporary data set (TESTCAT.DSNDBC.TEMPDB.TMPDBASE.I0001.A001) with DSN1COPY. This step also uses the CHECK option to check the table space for page integrity errors. After DSN1COPY with the CHECK option ensures that no errors exist, STEP4 restarts the table space for access to DB2. STEP5 runs DSN1CHKR on the temporary data set.

DSN1CHKR prints the chains, beginning with the pointers on the RID option in the MAP (maintenance analysis procedure) parameter. In this example, the first pointer is on page 000002, at an offset of 6 bytes from record 1. The second pointer is on page 00000B, at an offset of 6 bytes from record 1.

The RIDs in STEP5 are for instruction only.

```
//YOUR JOBCARD
//*
//JOBCAT  DD DSNAME=DSNCAT1.USER.CATALOG,DISP=SHR
//STEP1    EXEC PGM=IDCAMS
//*********************************************************************
//*         ALLOCATE A TEMPORARY DATA SET FOR SYSDBASE                *
//*********************************************************************
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN    DD *
   DELETE                                                      -
        (TESTCAT.DSNDBC.TEMPDB.TMPDBASE.I0001.A001)            -
          CATALOG(DSNCAT)
   DEFINE CLUSTER                                              -
        ( NAME(TESTCAT.DSNDBC.TEMPDB.TMPDBASE.I0001.A001)      -
          NONINDEXED                                           -
          REUSE                                                -
          CONTROLINTERVALSIZE(4096)                            -
          VOLUMES(XTRA02)                                      -
          RECORDS(783 783)                                     -
          RECORDSIZE(4089 4089)                                -
          SHAREOPTIONS(3 3) )                                  -
   DATA                                                        -
        ( NAME(TESTCAT.DSNDBD.TEMPDB.TMPDBASE.I0001.A001))     -
          CATALOG(DSNCAT)
/*
//STEP2    EXEC PGM=IKJEFT01,DYNAMNBR=20
//*********************************************************************
//*              STOP DSNDB06.SYSDBASE                                *
//*********************************************************************
//STEPLIB  DD DSN=prefix.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
     -STOP DB(DSNDB06) SPACENAM(SYSDBASE)
 END
/*
//STEP3    EXEC PGM=DSN1COPY,PARM=(CHECK)
//*********************************************************************
//*              CHECK SYSDBASE AND RUN DSN1COPY                      *
//*********************************************************************
//STEPLIB  DD DSN=prefix.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSN=DSNCAT.DSNDBC.DSNDB06.SYSDBASE.I0001.A001,DISP=SHR
//SYSUT2   DD DSN=TESTCAT.DSNDBC.TEMPDB.TMPDBASE.I0001.A001,DISP=SHR
/*
```

Figure 127. Sample JCL for running DSN1CHKR on a temporary data set (Part 1 of 2)

```
//STEP4    EXEC PGM=IKJEFT01,DYNAMNBR=20
//*******************************************************************
//*              START DSNDB06.SYSDBASE                             *
//*******************************************************************
//STEPLIB  DD DSN=prefix.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
     -START DB(DSNDB06) SPACENAM(SYSDBASE)
 END
/*//STEP5    EXEC  PGM=DSN1CHKR,PARM='MAP=RID(00000201,06,00000B01,06)',
//         COND=(4,LT)
//*******************************************************************
//*              CHECK LINKS OF SYSDBASE                            *
//*******************************************************************
//STEPLIB  DD DSN=prefix.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSN=TESTCAT.DSNDBC.TEMPDB.TMPDBASE.I0001.A001,DISP=SHR
/*
```

Figure 127. Sample JCL for running DSN1CHKR on a temporary data set (Part 2 of 2)

Example 2: Running DSN1CHKR on a table space. In the sample JCL in Figure 128, STEP1 stops database DSNDB06 with the STOP DATABASE command. STEP2 runs DSN1CHKR on the target table space; the output from this utility job is identical to the output in Example 1. STEP3 restarts the database with the START DATABASE command.

```
//YOUR JOBCARD
//*
//STEP1    EXEC PGM=IKJEFT01,DYNAMNBR=20
//*******************************************************************
//*                        EXAMPLE 2                                *
//*                                                                 *
//*                 STOP DSNDB06.SYSDBASE                           *
//*******************************************************************
//STEPLIB  DD DSN=prefix.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
     -STOP DB(DSNDB06) SPACENAM(SYSDBASE)
 END
/*
```

Figure 128. Sample JCL for running DSN1CHKR on a stopped table space. (Part 1 of 2)

```
//STEP2    EXEC  PGM=DSN1CHKR,PARM='MAP=RID(00000201,06,00000B01,06)',
//         COND=(4,LT)
//*******************************************************************
//*              CHECK LINKS OF SYSDBASE                            *
//*******************************************************************
//STEPLIB  DD DSN=prefix.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSN=DSNCAT.DSNDBD.DSNDB06.SYSDBASE.I0001.A001,DISP=SHR
/*
//STEP3    EXEC PGM=IKJEFT01,DYNAMNBR=20
//*******************************************************************
//*              RESTART DSNDB06.SYSDBASE                           *
//*******************************************************************
//STEPLIB  DD DSN=prefix.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
     -START DB(DSNDB06) SPACENAM(SYSDBASE)
 END
/*
```

Figure 128. Sample JCL for running DSN1CHKR on a stopped table space. (Part 2 of 2)

DSN1CHKR output

One intended use of this utility is to aid in determining and correcting system problems. When diagnosing DB2, you might need to refer to licensed documentation to interpret output from this utility. For more information about diagnosing problems, see *DB2 Diagnosis Guide and Reference*.

Chapter 39. DSN1COMP

DSN1COMP estimates space savings that are to be achieved by DB2 data compression in table spaces. For more information regarding ESA data compression, see Part 5 (Volume 2) of *DB2 Administration Guide*.

You can run this utility on the following types of data sets that contain uncompressed data:

- DB2 full image copy data sets
- VSAM data sets that contain DB2 table spaces
- Sequential data sets that contain DB2 table spaces (for example, DSN1COPY output)

You cannot run DSN1COMP on concurrent copies.

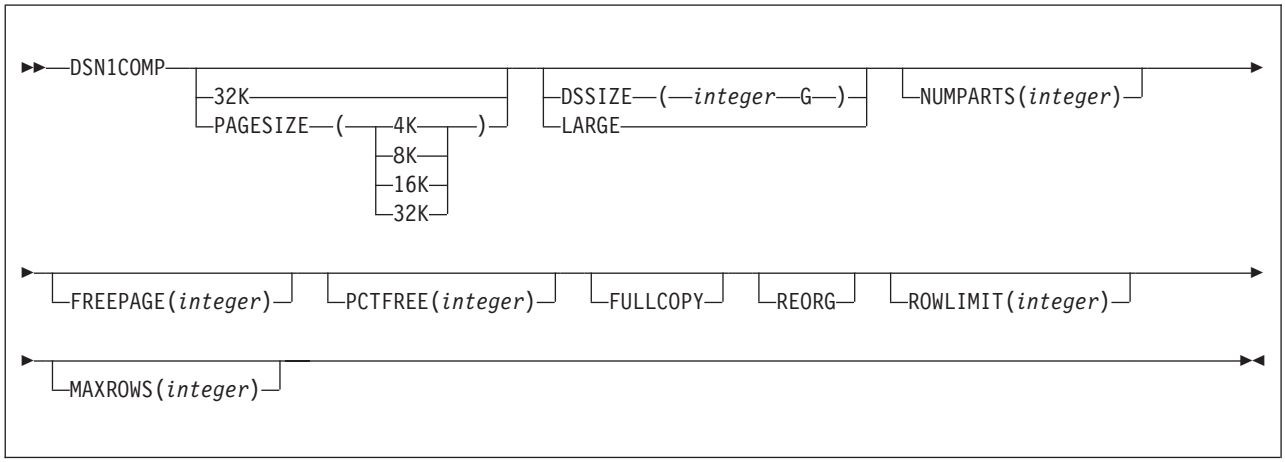
DSN1COMP does not estimate savings for data sets that contain LOB table spaces or for index spaces.

The following topics provide additional information:

- "Syntax and options of the DSN1COMP control statement"
- "Before running DSN1COMP" on page 719
- "Using DSN1COMP to estimate space savings from DB2 data compression" on page 721
- "Sample DSN1COMP control statements" on page 723
- "DSN1COMP output" on page 725

Syntax and options of the DSN1COMP control statement

DSN1COMP syntax diagram



Option descriptions

To run DSN1COMP, specify one or more of the following parameters on the EXEC statement to run DSN1COMP. If you specify more than one parameter, separate each parameter by a comma. You can specify parameters in any order.

32K Specifies that the input data set, SYSUT1, has a 32-KB page size. If you specify this option and the SYSUT1 data set does not have a 32-KB page size, DSN1COMP might produce unpredictable results.

The recommended option for performance is **PAGESIZE(32K)**.

PAGESIZE Specifies the page size of the input data set that is defined by SYSUT1. Available page size values are 4K, 8K, 16K, or 32K. If you specify an incorrect page size, DSN1COMP might produce unpredictable results.

If you omit PAGESIZE, DSN1COMP tries to determine the page size from the input data set. DB2 issues an error message if DSN1COMP cannot determine the input page size. This might happen if the header page is not in the input data set, or if the page size field in the header page contains an invalid page size.

DSSIZE(*integer* **G)**
Specifies the data set size, in gigabytes, for the input data set. If you omit DSSIZE, DB2 assumes that the input data set size is 2 GB.

integer must match the DSSIZE value that was specified when the table space was defined.

If you omit DSSIZE and the data set is not one of the default sizes, the results from DSN1COMP are unpredictable.

LARGE Specifies that the input data set is a table space that was defined with the LARGE option. If you specify LARGE, DB2 assumes that the data set has a 4-GB boundary.

The recommended method of specifying a table space defined with LARGE is **DSSIZE(4G)**.

If you omit the LARGE or DSSIZE(4G) option when it is needed, or if you specify LARGE for a table space that was not defined with the LARGE option, the results from DSN1COMP are unpredictable.

NUMPARTS(*integer***)**
Specifies the number of partitions that are associated with the input data set. Valid specifications range from 1 to 4096. If you omit NUMPARTS or specify it as 0, DSN1COMP assumes that your input file is not partitioned. If you specify a number greater than 64, DSN1COMP assumes that the data set is for a partitioned table space that was defined with the LARGE option, even if the LARGE keyword is not specified.

DSN1COMP cannot always validate the NUMPARTS parameter. If you specify it incorrectly, DSN1COMP might produce unpredictable results.

DSN1COMP terminates and issues message DSN1946I when it encounters an image copy that contains multiple partitions; a compression report is issued for the first partition.

FREEPAGE(*integer***)**
Specifies how often to leave a page of free space when calculating the percentage of saved pages. You must specify an integer in the range 0 to 255. If you specify 0, no pages are included as free space when DSN1COMP reports the percentage of pages saved.

Otherwise, one free page is included after every *n* pages, where *n* is the specified integer. The **default** is **0**.

Specify the same value that you specify for the FREEPAGE option of the SQL statement CREATE TABLESPACE or ALTER TABLESPACE.

PCTFREE(*integer***)**
Indicates what percentage of each page to leave as free space when calculating the percentage of pages saved. You must specify an integer in the range 0 to 99. When calculating the savings, DSN1COMP allows for at least *n* percent of free space for each page, where *n* is the specified integer. The **default** is **5**.

Specify the same value that you specify for the PCTFREE option of the SQL statement CREATE TABLESPACE or ALTER TABLESPACE.

FULLCOPY Specifies that a DB2 full image copy (not a DFSMSdss concurrent copy) of your data is to be used as input. Omitting this parameter when the input is a full image copy can cause error messages or unpredictable results. If this data is partitioned, also specify the NUMPARTS parameter to identify the number of partitions.

REORG Provides an estimate of compression savings that are comparable to the savings that the REORG utility would achieve. If this keyword is not specified, the results are similar to the compression savings that the LOAD utility would achieve.

ROWLIMIT(*integer***)**
Specifies the maximum number of rows to evaluate in order to provide the compression estimate. This option prevents DSN1COMP from examining every row in the input data set. Valid specifications range from 1 to 99000000.

Use this option to limit the elapsed time and processor time that DSN1COMP requires. An analysis of the first 5 to 10 MB of a table space provides a fairly representative sample of the table space for estimating compression savings. Therefore, specify a ROWLIMIT value that restricts DSN1COMP to the first 5 to 10 MB of the table space. For example, if the row length of the table space is 200 bytes, specifying ROWLIMIT(50000) causes DSN1COMP to analyze approximately 10 MB of the table space.

MAXROWS(*integer***)**
Specifies the maximum number of rows that DSN1COMP is to consider when calculating the percentage of pages saved. You must specify an integer in the range 1 to 255. The **default** is **255**.

Specify the same value that you specify for the MAXROWS option of the SQL statement CREATE TABLESPACE or ALTER TABLESPACE.

Before running DSN1COMP

If you run DSN1COMP on a segmented table space, you must first query the SYSTABLEPART catalog table to determine the current instance qualifier, which is stored in the IPREFIX column. You can then use the current instance qualifier to code the data set name in the JCL. The following sample shows an example of such a query.

```
SELECT DBNAME, TSNAME, PARTITION, IPREFIX
      FROM SYSIBM.SYSTABLEPART
      WHERE DBNAME = 'DBMC0731' AND TSNAME = 'TPMC0731'
      ORDER BY TSNAME, PARTITION;
```

The preceding query produces the following result:

```
  +-----------------------------------------+
  |  DBNAME  |  TSNAME  | PARTITION | IPREFIX |
  +-----------------------------------------+
1_| DBMC0731 | TPMC0731 |         1 | J       |
2_| DBMC0731 | TPMC0731 |         2 | J       |
3_| DBMC0731 | TPMC0731 |         3 | J       |
4_| DBMC0731 | TPMC0731 |         4 | J       |
5_| DBMC0731 | TPMC0731 |         5 | J       |
  +-----------------------------------------+
```

Figure 129. Result from query on the SYSTABLEPART catalog table to determine the value in the IPREFIX column

The preceding output provides the current instance qualifier (J), which can be used to code the data set name in the DSN1COMP JCL as follows.

```
//STEP1         EXEC PGM=DSN1COMP
//SYSUT1        DD DSN=vcatname.DSNDBC.DBMC0731.J0001.A001,DISP=SHR
//SYSPRINT      DD AYAOUT=*
//SYSUDUMP      DD AYAOUT=*
```

Environment

Run DSN1COMP as a z/OS job.

You can run DSN1COMP even when the DB2 subsystem is not operational. Before you use DSN1COMP when the DB2 subsystem is operational, issue the DB2 STOP DATABASE command. Issuing the STOP DATABASE command ensures that DB2 has not allocated the DB2 data sets.

Do not run DSN1COMP on table spaces in DSNDB01, DSNDB06, or DSNDB07.

Authorization required

DSN1COMP does not require authorization. However, if any of the data sets is RACF-protected, the authorization ID of the job must have RACF authority.

Control statement

Create the utility control statement for the DSN1COMP job. See “Syntax and options of the DSN1COMP control statement” on page 717 for DSN1COMP syntax and option descriptions.

Required data sets: DSN1COMP uses the following data definition (DD) statements:

SYSPRINT Defines the data set that contains output messages from DSN1COMP and all hexadecimal dump output.

SYSUT1 Defines the input data set, which can be a sequential data set or a VSAM data set.

Specify the disposition for this data set as OLD (DISP=OLD) to ensure that it is not in use by DB2. Specify the disposition for this

data set as SHR (DISP=SHR) only in circumstances where the DB2 STOP DATABASE command does not work.

The requested operation takes place only for the specified data set. In the following situations, you must specify the correct data set.
- The input data set belongs to a linear table space.
- The index space is larger than 2 GB.
- The table space or index space is a partitioned space.

If running the online REORG utility with the FASTSWITCH option, verify the data set name before running the DSN1COMP utility. The fifth-level qualifier in the data set name alternates between I0001 and J0001 when using FASTSWITCH. Specify the correct fifth-level qualifier in the data set name to successfully execute the DSN1COMP utility. To determine the correct fifth-level qualifier, query the IPREFIX column of SYSIBM.SYSTABLEPART for each data partition or the IPREFIX column of SYSIBM.SYSINDEXPART for each index partition. If the object is not partitioned, use zero as the value for the PARTITION column in your query.

Recommendation

Before using DSN1COMP, be sure that you know the page size and data set size (DSSIZE) for the table space. Use the following query on the DB2 catalog to get the information you need, in this example for table 'DEPT':

```
SELECT T.CREATOR,T.NAME,S.NAME AS TABLESPACE,S.PARTITIONS,S.PGSIZE,
      CASE S.DSSIZE
      WHEN 0 THEN
                  CASE WHEN S.TYPE = 'O' THEN 4194304
                  ELSE
                       CASE WHEN S.PARTITIONS > 254 THEN
                            CASE WHEN S.PGSIZE = 4  THEN  4194304
                                 WHEN S.PGSIZE = 8  THEN  8388608
                                 WHEN S.PGSIZE = 16 THEN 16777216
                                 WHEN S.PGSIZE = 32 THEN 33554432
                            ELSE NULL
                            END
                            WHEN S.PARTITIONS > 64 THEN 4194304
                            WHEN S.PARTITIONS > 32 THEN 1048576
                            WHEN S.PARTITIONS > 16 THEN 2097152
                            WHEN S.PARTITIONS > 0  THEN 4194304
                       ELSE 2097152
                       END
                  END
      ELSE S.DSSIZE
      END
      AS DSSIZE
      FROM SYSIBM.SYSTABLES T,
           SYSIBM.SYSTABLESPACE S
      WHERE
           T.NAME = 'DEPT' AND
           T.TSNAME = S.NAME;
```

Using DSN1COMP to estimate space savings from DB2 data compression

This section describes the following tasks that are associated with running the DSN1COMP utility:

"The effect of the REORG option on compression savings estimates" on page 722

"Free space in compression calculations" on page 722

"The effect of running DSN1COMP on a table space with identical rows" on page 723

The effect of the REORG option on compression savings estimates

If you run DSN1COMP with the REORG option on small data sets, the resulting estimates might vary greatly from the estimates that are produced without the default REORG option. Alternatively, if you run DSN1COMP and specify a small number (*n*) for ROWLIMIT, the estimates might vary greatly from the estimates that are produced without REORG.

DNS1COMP does not try to convert data to the latest version before it compresses rows and derives a savings estimate.

Without the REORG option, DSN1COMP uses the first *n* rows to fill the compression dictionary. DSN1COMP processes the remaining rows to provide the compression estimate. If the number of rows that are used to build the dictionary is a significant percentage of the data set rows, little savings result. With the REORG option, DSN1COMP processes all the rows, including those that are used to build the dictionary, which results in greater compression.

Free space in compression calculations

The DSN1COMP utility's compression estimates take into account the PCTFREE and FREEPAGE options. If you use different PCTFREE or FREEPAGE values than those that were created with the input table space, you get a different value for **noncmppages**. DSN1COMP reports this value in message DSN1940I, as shown in the example output in Figure 130 on page 723.

```
DSN1999I START OF DSN1COMP FOR JOB TST512A  STEP1
DSN1998I INPUT DSNAME = FUFOU237.TSP32K                              , SEQ
DSN1944I DSN1COMP INPUT PARAMETERS
             512  DICTIONARY SIZE USED
              30  FREEPAGE VALUE USED
              45  PCTFREE VALUE USED
                  NO ROWLIMIT WAS REQUESTED
                  ESTIMATE BASED ON DB2 LOAD METHOD

DSN1940I DSN1COMP COMPRESSION REPORT
           1,289  KB WITHOUT COMPRESSION
             717  KB WITH COMPRESSION
              44  PERCENT OF THE BYTES WOULD BE SAVED

             176  ROWS SCANNED TO BUILD DICTIONARY
          20,000  ROWS SCANNED TO PROVIDE COMPRESSION ESTIMATE
             512  DICTIONARY ENTRIES

               1  DICTIONARY PAGES REQUIRED
             147  PAGES REQUIRED WITHOUT COMPRESSION
             148  PAGES REQUIRED WITH COMPRESSION
               0  PERCENT OF THE DB2 DATA PAGES WOULD BE SAVED

   *** DETAIL REPORT OF FREQUENCIES AND AVERAGES ***

     1 CHILD CHARACTER WAS COMPARED                          566,764  TIMES
     2 CHILD CHARACTERS WERE COMPARED                        182,026  TIMES
     3 CHILD CHARACTERS WERE COMPARED                         10,300  TIMES
     5 CHILD CHARACTERS WERE COMPARED                          1,129  TIMES
     TOTAL ALPHABET NODE COMPARISONS                         528,139  TIMES

                 967,361  CHILD COMPARISONS IN THE SIBLING LISTS
                 760,219  SEARCHES IN THE SIBLING LISTS
                     1.2  AVERAGE NUMBER OF COMPARISONS PER SEARCH
              60  BYTES FOR AVERAGE UNCOMPRESSED ROW LENGTH
              39  BYTES FOR AVERAGE COMPRESSED ROW LENGTH

             263  IS THE DATABASE ID (DBID)
               2  IS THE PAGESET  ID (PSID)
```

Figure 130. Example DSN1COMP output

The effect of running DSN1COMP on a table space with identical rows

If you run DSN1COMP on a table space in which the data is the same for all rows, message DSN1941I is issued. In this case, DSN1COMP does not compute any statistics.

Sample DSN1COMP control statements

Example 1: Estimating space savings from data compression for a full image copy. The following statement specifies that the DSN1COMP utility is to report the estimated space savings that are to be achieved by compressing the full image copy that is identified by the SYSUT1 DD statement. In this statement, the DSN option specifies the data set name of the image copy that is to be used as input. The fifth qualifier in the data set name can be either I0001 or J0001. This example uses I0001. Note that because the input is a full image copy, the FULLCOPY option must be specified.

```
//jobname JOB acct information
//COMPEST  EXEC PGM=DSN1COMP,PARM='FULLCOPY'
//STEPLIB  DD DSN=prefix.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSABEND DD SYSOUT=A
//SYSUT1   DD DSN=DSNCAT.DSNDBC.DB254A.TS254A.I0001.A001,DISP=SHR
```

Example 2: Providing intended free space when estimating space savings. In the sample statements in Figure 131, STEP1 specifies that DSN1COMP is to report the estimated space savings that are to be achieved by compressing the data in the data set that is identified by the SYSUT1 DD statement, DSNC810.DSNDBD.DB254SP4.TS254SP4.I0001.A00. When calculating these estimates, DSN1COMP considers the values passed by the PCTFREE and FREEPAGE options. The PCTFREE value indicates that 20% of each page is to be left as free space. The FREEPAGE value indicates that every fifth page is to be left as free space. This value must be the same value that you specified for the FREEPAGE option of the SQL statement CREATE TABLESPACE or ALTER TABLESPACE.

STEP2 specifies that DSN1COMP is to report the estimated space savings that are to achieved by compressing the data in the data set that is identified by the SYSUT1 DD statement, DSNC810.DSNDBD.DB254SP4.TS254SP4.I0001.A0001. When providing the compression estimate, DSN1COMP is to evaluate no more than 20 000 rows, as indicated by the ROWLIMIT option. Specifying the maximum number of rows to evaluate limits the elapsed time and processor time that DSN1COMP requires.

```
//DSN1COMP JOB  MSGLEVEL=(1,1),CLASS=A,MSGCLASS=A,REGION=3000K,
//         USER=SYSADM,PASSWORD=SYSADM
/*ROUTE PRINT STLXXXX.USERID
//STEP1    EXEC PGM=DSN1COMP,PARM='PCTFREE(20),FREEPAGE(5)'
//STEPLIB  DD DSN=prefix.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSDUMP  DD SYSOUT=A
//SYSABEND DD SYSOUT=A
//SYSUT1   DD DSN=DSNC810.DSNDBD.DB254SP4.TS254SP4.I0001.A001,DISP=SHR
/*
//STEP2    EXEC PGM=DSN1COMP,PARM='ROWLIMIT(20000)'
//STEPLIB  DD DSN=prefix.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSDUMP  DD SYSOUT=A
//SYSABEND DD SYSOUT=A
//SYSUT1   DD DSN=DSNC810.DSNDBD.DB254SP4.TS254SP4.I0001.A001,DISP=SHR
/*
//
```

Figure 131. Example DSN1COMP statements with PCTFREE, FREEPAGE, and ROWLIMIT options

Example 3: Estimating space savings that are comparable to what the REORG utility would achieve. The statement in Figure 132 on page 725 specifies that DSN1COMP is to report the estimated space savings that are to be achieved by compressing the data in the data set that is identified by the SYSUT1 DD statement, DSNCAT.DSNDBD.DBJT0201.TPJTO201.I0001.A254. This input data set is a table space that was defined with the LARGE option and has 254 partitions, as indicated by the DSN1COMP options LARGE and NUMPARTS.

The REORG option indicates that DSN1COMP is to provide an estimate of compression savings that is comparable to the savings that the REORG utility would achieve, rather than what the LOAD utility would achieve.

When calculating these estimates, DSN1COMP considers the values passed by the PCTFREE and FREEPAGE options. The PCTFREE value indicates that 30% of each page is to be left as free space. The FREEPAGE value indicates that every thirtieth page is to be left as free space. This value must be the same value that you specified for the FREEPAGE option of the SQL statement CREATE TABLESPACE or ALTER TABLESPACE. DSN1COMP is to evaluate no more than 20 000 rows, as indicated by the ROWLIMIT option.

```
//STEP2  EXEC PGM=DSN1COMP,
//             PARM='LARGE,PCTFREE(30),FREEPAGE(30),NUMPARTS(
//             254),REORG,ROWLIMIT(1000)'
//STEPLIB  DD DSN='USER.TESTLIB',DISP=SHR
//         DD DSN='DB2A.SDSNLOAD',DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSDUMP  DD SYSOUT=A
//SYSABEND DD SYSOUT=A
//SYSUT1   DD DSN=DSNCAT.DSNDBD.DBJT0201.TPJT0201.I0001.A254,DISP=SHR
//SYSUT2   DD SYSOUT=A
/*
```

Figure 132. Example DSN1COMP statement with the LARGE, PCTFREE, FREEPAGE, NUMPARTS, REORG, and ROWLIMIT options.

DSN1COMP output

This section contains examples of output that is generated by the DSN1COMP utility.

Message DSN1941

If you receive this message, use a data set with more rows as input, or specify a larger ROWLIMIT.

Sample DSN1COMP report

Figure 133 shows a sample of the output that DSN1COMP generates.

```
DSN1940I DSN1COMP COMPRESSION REPORT
             301  KB WITHOUT COMPRESSION
             224  KB WITH COMPRESSION
              25  PERCENT OF THE BYTES WOULD BE SAVED

           1,975  ROWS SCANNED TO BUILD DICTIONARY
           4,665  ROWS SCANNED TO PROVIDE COMPRESSION ESTIMATE
           4,096  DICTIONARY ENTRIES

              81  BYTES FOR AVERAGE UNCOMPRESSED ROW LENGTH
              52  BYTES FOR AVERAGE COMPRESSED ROW LENGTH

              16  DICTIONARY PAGES REQUIRED
             110  PAGES REQUIRED WITHOUT COMPRESSION
              99  PAGES REQUIRED WITH COMPRESSION
              10  PERCENT OF THE DB2 DATA PAGES WOULD BE SAVED
```

Figure 133. Sample DSN1COMP report

Chapter 40. DSN1COPY

With the DSN1COPY stand-alone utility, you can copy:
- DB2 VSAM data sets to sequential data sets
- DSN1COPY sequential data sets to DB2 VSAM data sets
- DB2 image copy data sets to DB2 VSAM data sets
- DB2 VSAM data sets to other DB2 VSAM data sets
- DSN1COPY sequential data sets to other sequential data sets

Note: A DB2 VSAM data set is a single piece of a nonpartitioned table space or index, or a single partition of a partitioned table space or index. The input must be a single z/OS sequential or VSAM data set. Concatenation of input data sets is not supported.

Using DSN1COPY, you can also print hexadecimal dumps of DB2 data sets and databases, check the validity of data or index pages (including dictionary pages for compressed data), translate database object identifiers (OBIDs) to enable moving data sets between different systems, and reset to 0 the log RBA that is recorded in each index page or data page.

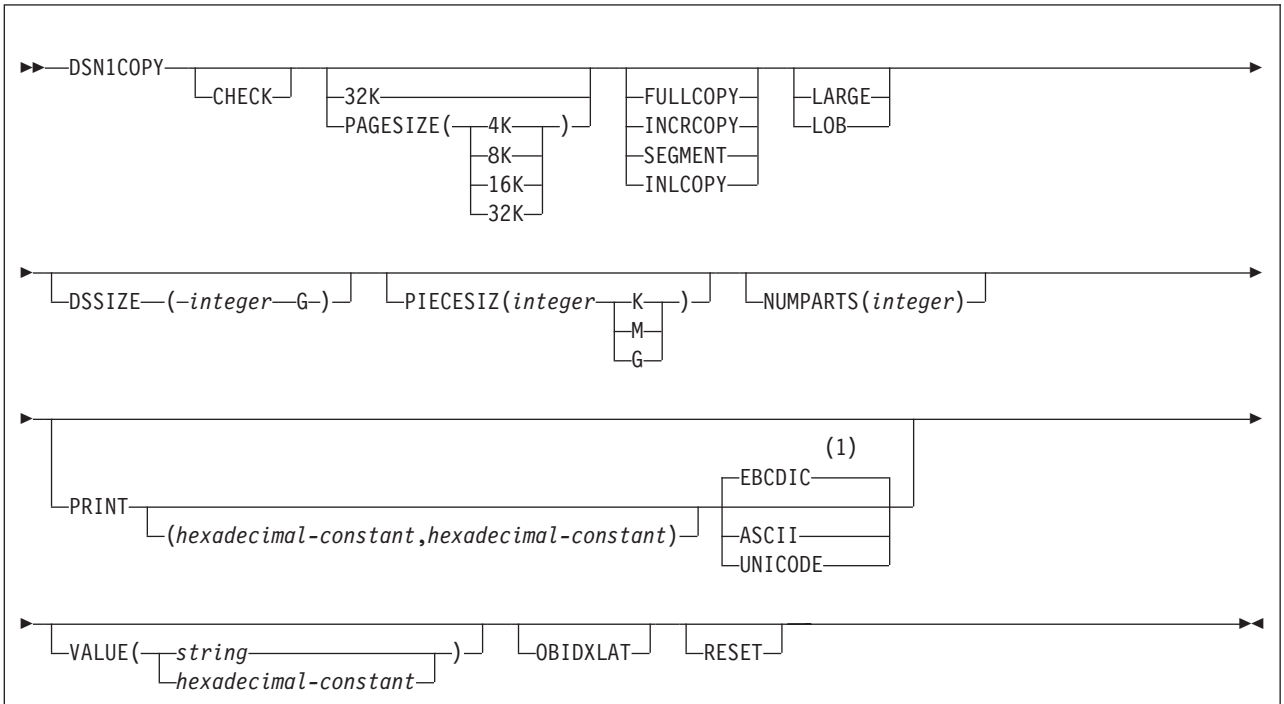
You cannot run DSN1COPY on concurrent copies.

You can use the DSN1COPY utility on LOB table spaces by specifying the LOB keyword and omitting the SEGMENT and INLCOPY keywords.

The following topics provide additional information:
- "Syntax and options of the DSN1COPY control statement" on page 728
- "Before running DSN1COPY" on page 733
- "Using DSN1COPY to copy data sets" on page 740
- "Sample DSN1COPY control statements" on page 743
- "DSN1COPY output" on page 745

Syntax and options of the DSN1COPY control statement

DSN1COPY syntax diagram



Notes:

1 EBCDIC is not necessarily the default if the first page of the input data set is a header page. If the first page is a header page, DSN1COPY uses the format information in the header page as the default format.

Option descriptions

To run DSN1COPY, specify one or more of the following parameters on the EXEC statement. If you specify more than one parameter, separate each parameter by a comma. You can specify parameters in any order.

CHECK Checks each page from the SYSUT1 data set for validity. The validity checking operates on one page at a time and does not include any cross-page checking. If an error is found, a message is issued describing the type of error, and a dump of the page is sent to the SYSPRINT data set. If you do not receive any messages, no errors were found. If more than one error exists in a given page, the check identifies only the first of the errors. However, the entire page is dumped. DSN1COPY does not check system pages for validity.

32K Specifies that the SYSUT1 data set has a 32-KB page size. If you specify this option and the SYSUT1 data set does not have a 32-KB page size, DSN1COPY might produce unpredictable results.

The recommended option for performance is **PAGESIZE(32K)**.

PAGESIZE Specifies the page size of the input data set that is defined by

SYSUT1. Available page size values are 4K, 8K, 16K, or 32K. If you specify an incorrect page size, DSN1COPY might produce unpredictable results.

If you do not specify the page size, DSN1COPY tries to determine the page size from the input data set if the first page of the input data set is a header page. DB2 issues an error message if DSN1COPY cannot determine the input page size. This might happen if the header page is not in the input data set, or if the page size field in the header page contains an invalid page size.

FULLCOPY Specifies that a DB2 full image copy (not a DFSMSdss concurrent copy) of your data is to be used as input. If this data is partitioned, specify NUMPARTS to identify the total number of partitions. If you specify FULLCOPY without NUMPARTS, DSN1COPY assumes that your input file is not partitioned.

Specify FULLCOPY when using a full image copy as input. Omitting the parameter can cause error messages or unpredictable results.

The FULLCOPY parameter requires SYSUT2 (output data set) to be either a DB2 VSAM data set or a DUMMY data set.

INCRCOPY Specifies that an incremental image copy of the data is to be used as input. DSN1COPY with the INCRCOPY parameter updates existing data sets; do not redefine the existing data sets. INCRCOPY requires that the output data set (SYSUT2) be a DB2 VSAM data set.

Before you apply an incremental image copy to your data set, you must first apply a full image copy to the data set by using the FULLCOPY parameter. Make sure that you apply the full image copy in a separate execution step because you receive an error message if you specify both the FULLCOPY and the INCRCOPY parameters in the same step. Then, apply each incremental image copy in a separate step, starting with the oldest incremental image copy.

Specifying neither FULLCOPY nor INCRCOPY implies that the input is not image copy data sets. Therefore, only a single output data set is used.

SEGMENT Specifies that you want to use a segmented table space as input to DSN1COPY. Pages with all zeros in the table space are copied, but no error messages are issued. You cannot specify FULLCOPY or INCRCOPY if you specify SEGMENT.

If you are using DSN1COPY with the OBIDXLAT to copy a DB2 data set to another DB2 data set, the source and target table spaces must have the same SEGSIZE attribute.

You cannot specify the SEGMENT option with the LOB parameter.

INLCOPY Specifies that the input data is an inline copy data set.

You cannot specify the INLCOPY option with the LOB parameter.

DSSIZE(*integer* **G)**
Specifies the data set size, in gigabytes, for the input data set. If you omit the DSSIZE keyword or the LARGE keyword, DSN1COPY assumes the appropriate default input data set size

that is listed in Table 140.

Table 140. Default input data set sizes

Object	Default input data set size (in GB)
Non-LOB linear table space or index	2
LOB	4
Partitioned table space or index with NUMPARTS = 1-16	4
Partitioned table space or index with NUMPARTS = 17-32	2
Partitioned table space or index with NUMPARTS = 33-64	1
Partitioned table space or index with NUMPARTS >64	4

integer must match the DSSIZE value that was specified when the table space was defined.

If you omit DSSIZE and the data set is not one of the default sizes, the results from DSN1COPY are unpredictable.

If you specify DSSIZE, you cannot specify LARGE.

LARGE Specifies that the input data set is a table space that was defined with the LARGE option, or an index on such a table space. If you specify the LARGE keyword, DB2 assumes that the data set has a 4-GB boundary. The recommended method of specifying a table space that was defined with the LARGE option is **DSSIZE(4G)**.

If you omit the LARGE or DSSIZE(4G) option when it is needed, or if you specify LARGE for a table space that was not defined with the LARGE option, the results from DSN1COPY are unpredictable.

If you specify LARGE, you cannot specify LOB or DSSIZE.

LOB Specifies that SYSUT1 data set is a LOB table space. Empty pages in the table space are copied, but no error messages are issued. You cannot specify the SEGMENT and INLCOPY options with the LOB parameter.

DSN1COPY attempts to determine if the input data set is a LOB data set. If it can be clearly verified that the LOB option is specified, but the data set is not a LOB data set, or that the LOB option is omitted for a data set that is a LOB data set, DSN1COPY issues an error message and terminates. Otherwise, if the LOB option isn't specified or omitted correctly the results of DSN1COPY are unpredictable.

If you specify LOB, you cannot specify LARGE.

NUMPARTS(*integer***)**
Specifies the total number of partitions that are associated with the data set that you are using as input or whose page range you are printing. When you use DSN1COPY to copy a data-partitioned secondary index, specify the number of partitions in the index.

integer can range from 1 to 4096.

DSN1COPY uses this value to calculate the size of its output data sets and to help locate the first page in a range that is to be printed. If you omit NUMPARTS or specify it as 0, DSN1COPY will get the NUMPARTS value from the header page if possible, otherwise DSN1COPY will assume that your input is not partitioned. If you specify a number greater than 64, DSN1COPY assumes that the data set is for a partitioned table space that was defined with the LARGE option, even if the LARGE keyword is not specified for DSN1COPY.

If you specify the number of partitions incorrectly, DSN1COPY can copy the data to the wrong data sets, return an error message indicating that an unexpected page number was encountered, or fail to allocate the data sets correctly. In the last case, a VSAM PUT error might be detected, resulting in a request parameter list (RPL) error code of 24.

PRINT(*hexadecimal-constant,hexadecimal-constant***)**

Causes the SYSUT1 data set to be printed in hexadecimal format on the SYSPRINT data set. You can specify the PRINT parameter with or without the page range specifications (*hexadecimal-constant,hexadecimal-constant*). If you do not specify a range, all pages of the SYSUT1 are printed. If you want to limit the range of pages that are printed, indicate the beginning and ending page. If you want to print a single page, supply only that page number. In either case, your range specifications must be from one to eight hexadecimal characters in length.

The following example shows how you code the PRINT parameter if you want to begin printing at page X'2F0' and stop at page X'35C':

```
PRINT(2F0,35C)
```

Because the CHECK and RESET options and the copy function run independently of the PRINT range, these options apply to the entire input file, regardless of whether a range of pages is being printed.

You can indicate the format of the row data in the PRINT output by specifying EBCDIC, ASCII, or UNICODE. For an example of the output that is affected by these options, see the DSN1PRNT FORMAT output in Figure 142 on page 772.

EBCDIC

Indicates that the row data in the PRINT output is to be displayed in EBCDIC. The **default** is **EBCDIC** if the first page of the input data set is not a header page.

If the first page is a header page, DSN1COPY uses the format information in the header page as the default format. However, if you specify EBCDIC, ASCII, or UNICODE, that format overrides the format information in the header page. The unformatted header page dump is always displayed in EBCDIC, because most of the fields are in EBCDIC.

ASCII

Indicates that the row data in the PRINT output is to be displayed in ASCII. Specify ASCII when printing table spaces that contain ASCII data.

UNICODE
Indicates that the row data in the PRINT output is to be displayed in Unicode. Specify UNICODE when printing table spaces that contain Unicode data.

PIECESIZ(*integer***)**
Specifies the maximum piece size (data set size) for nonpartitioned indexes. The value that you specify must match the value that was specified when the nonpartitioning index was created or altered.

The defaults for PIECESIZ are 2G (2 GB) for indexes that are backed by non-large table spaces and 4G (4 GB) for indexes that are backed by table spaces that were defined with the LARGE option. This option is required if the piece size is not one of the default values. If PIECESIZ is omitted and the index is backed by a table space that was defined with the LARGE option, the LARGE option is required for DSN1COPY.

The subsequent keyword K, M, or G indicates the unit of the value that is specified in *integer*.

K Indicates that the *integer* value is to be multiplied by 1 KB to specify the maximum piece size in bytes. *integer* must be either 256 or 512.

M Indicates that the *integer* value is to be multiplied by 1 MB to specify the maximum piece size in bytes. *integer* must be a power of two, between 1 and 512.

G Indicates that the *integer* value is to be multiplied by 1 GB to specify the maximum piece size in bytes. *integer* must be 1, 2, or 4.

Valid values for piece size are:
- 1 MB or 1 GB
- 2 MB or 2 GB
- 4 MB or 4 GB
- 8 MB
- 16 MB
- 32 MB
- 64 MB
- 128 MB
- 256 KB or 256 MB
- 512 KB or 512 MB

VALUE Causes each page of the SYSUT1 input data set to be scanned for the character string that you specify in parentheses following the VALUE parameter. Each page that contains that character string is printed in the SYSPRINT data set. You can specify the VALUE parameter in conjunction with any of the other DSN1COPY parameters.

string can consist of 1 to 20 alphanumeric characters.

hexadecimal-constant can consist of 2 to 40 hexadecimal characters. Specify two apostrophe characters before and after the hexadecimal character string.

If you want to search your input file for the string '12345', your JCL should look similar to the following JCL:

```
//STEP1 EXEC PGM=DSN1COPY,PARM='VALUE(12345)'
```

If you want to search for the equivalent hexadecimal character string, your JCL should look similar to the following JCL:

```
//STEP1 EXEC PGM=DSN1COPY,PARM='VALUE(''F1F2F3F4F5'')'
```

OBIDXLAT Specifies that OBID translation must be done before the DB2 data set is copied. This parameter requires additional input from the SYSXLAT file by using the DD statements. DSN1COPY can translate only up to 1000 record OBIDs. If you specify OBIDXLAT, CHECK processing is performed, regardless of whether you specify the CHECK option.

RESET Causes the log RBAs in each index page or data page and the high-formatted page number in the header page to be reset to 0. If you specify this option, CHECK processing is performed, regardless of whether you specify the CHECK option.

Use RESET when the output file is used to build a DB2 table space that is to be processed on a DB2 subsystem with a different recovery log than the source subsystem. Failure to specify RESET in such a case can result in an abend during subsequent update activity. The abend reason code of 00C200C1 indicates that the specified RBA value is outside the valid range of the recovery log. A condition code of 0 indicates successful completion.

If you do not specify RESET when copying a table space from one DB2 system to another, a down-level ID check might result in abend reason code 00C2010D when the table space is accessed. For more information about down-level detection, see Part 4 (Volume 1) of *DB2 Administration Guide*.

Before running DSN1COPY

This section contains information that you should use before running DSN1COPY.

Attention: Do not use DSN1COPY in place of COPY for both backup and recovery. Improper use of DSN1COPY can result in unrecoverable damage and loss of data.

Environment

Execute DSN1COPY as a z/OS job when the DB2 subsystem is either active or not active.

If you execute DSN1COPY when DB2 is active, use the following procedure:

1. Start the table space as read-only by using START DATABASE.
2. Run the QUIESCE utility with the WRITE (YES) option to externalize all data pages and index pages.
3. Run DSN1COPY with DISP=SHR on the data definition (DD) statement.
4. Start the table space as read-write by using START DATABASE to return to normal operations.

Authorization required

DSN1COPY does not require authorization. However, if any of the data sets is RACF-protected, the authorization ID of the job must have RACF authority.

Control statement

Create the utility control statement for the DSN1COPY job. See "Syntax and options of the DSN1COPY control statement" on page 728 for DSN1COPY syntax and option descriptions.

Required data sets:
DSN1COPY uses the following data sets:

Input data set	Input to DSN1COPY. The DD name is SYSUT1.
Output data set	Output from DSN1COPY. The DD name is SYSUT2. Optional.
Message data set	Data set for output messages. The DD name is SYSPRINT.
OBIDXLAT data set	Data set that defines the OBID translation values. The DD name is SYSXLAT.

DSN1COPY uses the following DD statements:

SYSPRINT Defines the data set that contains output messages from the DSN1COPY program and all hexadecimal dump output.

SYSUT1 Defines the input data set. This data set can be a sequential data set that is created by the DSN1COPY or COPY utilities, or a VSAM data set.

Specify the data set's disposition as DISP=OLD to ensure that it is not in use by DB2. Specify the data set's disposition as DISP=SHR only when the DB2 STOP DATABASE command does not work.

The requested operation takes place only for the specified data set. If the input data set is a partitioned table space or index, ensure that you specify the NUMPARTS parameter and the correct data set. For example, to print a page range in the second partition of a four-partition table space, specify NUMPARTS(4) and the data set name of the second data set. This second data set is in the group of VSAM data sets, and the VSAM data set name is DSNCAT.DSNDBD.TESTDB.TS01.I0001.A002. The last qualifier (A002) represents the partition number 2. See "Sample DSN1COPY control statements" on page 743 for examples of VSAM data set names.

If running the online REORG utility with the FASTSWITCH option, verify the data set name before running the DSN1COPY utility. The fifth-level qualifier in the data set name alternates between I0001 and J0001 when using FASTSWITCH. Specify the correct fifth-level qualifier in the data set name to successfully execute the DSN1COPY utility. To determine the correct fifth-level qualifier, query the IPREFIX column of SYSIBM.SYSTABLEPART for each data partition or the IPREFIX column of SYSIBM.SYSINDEXPART for each index partition. If the object is not partitioned, use zero as the value for the PARTITION column in your query.

SYSUT2 Defines the output data set. This data set can be a sequential data set, a VSAM data set, or a DUMMY data set.

DSN1COPY assumes that the output data sets are empty (that is, the program adds the blocks) except when you specify INCRCOPY. Before you run DSN1COPY, define your VSAM output data sets as

REUSE. If you have not defined the data sets, you must redefine all VSAM output data sets you are restoring by using Access Method Services. Ensure that these data sets are empty before you run DSN1COPY.

You might want to specify a DUMMY SYSUT2 DD statement if you are dumping or checking pages.

To enable DB2 to obtain necessary information from the integrated catalog facility catalog when using VSAM data sets, do not code the unit-serial parameter and volume-serial parameter.

If running the online REORG utility with the FASTSWITCH option, verify the data set name before running the DSN1COPY utility. The fifth-level qualifier in the data set name alternates between I0001 and J0001 when using FASTSWITCH. Specify the correct fifth-level qualifier in the data set name to successfully execute the DSN1COPY utility.

SYSXLAT Defines for translation the DBIDs, OBIDs, presentation space ID (PSID), or ISOBIDs.

If you have dropped a table without a subsequent REORG of the table space, you must reorganize the source table space before running DSN1COPY with the OBIDXLAT option. This action removes any previously dropped records from the table space.

A non-numeric character must separate each record in the SYSXLAT file, and each record must contain a pair of decimal integers. The first integer of each record pertains to the source, and the second integer pertains to the target. The first record in the SYSXLAT file contains the source DBIDs and the target DBIDs; the values can range from -32767 to 65535. The second record contains the source and target PSIDs or ISOBIDs; the values can range from 0 to 32767. All subsequent records in the SYSXLAT data set are for table OBIDs. For an index, the SYSXLAT data set must contain the index fan set OBID, in addition to the DBID and ISOBID. Sample data in a SYSXLAT file follows (with an indication of how each record translates shown in parentheses):

```
260,280 (source DBID 260 translates to target DBID 280)
2,10    (source PSID 2 translates to target PSID 10)
3,55    (source table OBID 3 translates to target table OBID 55)
6,56    (source table OBID 6 translates to target table OBID 57)
7,57    (source table OBID 7 translates to target table OBID 57
```

To obtain the names, DBIDs, PSIDs, ISOBIDs, and OBIDs, run the DSNTEP2 sample application on both the source and target systems. The following SQL statements yield the preceding information.

The example for indexes yields output that is similar to the preceding example, but with an additional column of data.

Product-sensitive Programming Interface

For table spaces use the following statements:

```
SELECT DBID, PSID FROM SYSIBM.SYSTABLESPACE
   WHERE NAME='tablespace_name'
     AND DBNAME='database_name';
SELECT NAME, OBID FROM SYSIBM.SYSTABLES
   WHERE TSNAME='tablespace_name'
     AND CREATOR='creator_name';
```

For index spaces use the following statement:

```
SELECT DBID, ISOBID, OBID FROM SYSIBM.SYSINDEXES
   WHERE NAME='index_name'
     AND CREATOR='creator_name';
```

End of Product-sensitive Programming Interface

Several examples of using DSN1COPY follow:

- Create a backup copy of a DB2 data set:
  - SYSUT1: DB2-VSAM
  - SYSUT2: Sequential data set
- Restore a backup copy of a DB2 data set:
  - SYSUT1: DSN1COPY sequential data set
  - SYSUT2: DB2-VSAM
- Move a DB2 data set to another DB2 data set:
  - SYSUT1: DB2-VSAM
  - SYSUT2: DB2-VSAM
  - Parameters: OBIDXLAT, RESET
- Perform validity checking on a DB2 data set:
  - SYSUT1: DB2-VSAM
  - SYSUT2: DUMMY
  - Parameter: CHECK
- Perform validity checking on and print a DB2 data set:
  - SYSUT1: DB2-VSAM
  - SYSUT2: DUMMY
  - Parameters: CHECK, PRINT
- Restore a table space from a nonpartitioned image copy data set or page set:
  - SYSUT1: DB2 full image copy
  - SYSUT2: DB2-VSAM
  - Parameter: FULLCOPY
- Restore a table space from a partitioned image copy data or page set:
  - SYSUT1: DB2 full image copy
  - SYSUT2: DB2-VSAM
  - Parameters: FULLCOPY, NUMPARTS(*nn*)
- Perform RBA RESET on a DB2 data set:
  - SYSUT1: DB2-VSAM or DSN1COPY sequential data set
  - SYSUT2: DB2-VSAM
  - Parameter: RESET

Defining the input data set

The SYSUT1 data set can be any of the following types:

- A DB2 table space data set
- A DB2 index space data set
- A full image copy
- An incremental image copy
- A sequential data set that was previously created by DSN1COPY

Define SYSUT1 with DISP=OLD to ensure that DSN1COPY uses it exclusively. If SYSUT1 is a table space or index space, use the following procedure before using DSN1COPY:

1. Issue the following command to determine if the object is stopped:

   ```
   -DISPLAY DATABASE (database_name) SPACENAM(space_name) RESTRICT
   ```

2. If DB2 has not stopped the object, issue the following command to stop the object:

```
-STOP DATABASE (database_name) SPACENAME(space_name)
```

DB2 allows input of only one DSN1COPY data set. DB2 does not permit the input of concatenated data sets. For a table space that consists of multiple data sets, ensure that you specify the correct data set. For example, if you specify the CHECK option to validate pages of a partitioned table space's second partition, code the second data set of the table space for SYSUT1.

Defining the output data set

The SYSUT2 data set can be any of the following types:

- A sequential data set
- A DB2 table space data set
- A DB2 index space data set
- A DUMMY data set

Specify a DUMMY SYSUT2 DD statement if you are using DSN1COPY to check or dump a page. The table spaces and index spaces must either be empty or defined with VSAM REUSE. STOGROUP-defined table spaces and index spaces have the REUSE attribute, except when you are applying the INCRCOPY option

Naming the output data set

For your output data set to be useful, ensure that it has the same name as the data set that you are resetting.

- Method 1:
  1. Use DSN1COPY to copy your existing data set to a sequential data set. Specify this target data set as SYSUT1.
  2. If you defined your existing data set without the REUSE parameter, delete and redefine the data set. Specify your existing data set as SYSUT2.
- Method 2:
  1. Use your existing DB2 data set as the SYSUT1 specification, creating a new VSAM data set for SYSUT2.
  2. After completion of the reset operation, delete the data set that you specified as SYSUT1, and rename the SYSUT2 data set. Give SYSUT2 the name of the data set that you just deleted.

If you use full or incremental copies as input, specify the SYSUT2 data sets according to the following guidelines:

- **If SYSUT1 is an image copy of a single partition**, SYSUT2 must list the data set name for that partition of the table space. Specify the NUMPARTS parameter to identify the number of partitions in the entire table space.
- **If SYSUT1 is an image copy of an entire partitioned table space**, SYSUT2 must list the name of the table space's first data set. **Important:** All data sets in the partitioned table space must use the same fifth-level qualifier, I0001 or J0001, before DSN1COPY can run successfully on a partitioned table space. DSN1COPY allocates all of the target data sets. However, you must previously define the target data sets by using IDCAMS. Specify the NUMPARTS parameter to identify the number of partitions in the whole table space.
- **If SYSUT1 is an image copy of a nonpartitioned data set**, SYSUT2 should be the name of the actual output data set. Do not specify the NUMPARTS parameter because this parameter is only for partitioned table spaces.

- **If SYSUT1 is an image copy of all data sets in a linear table space with multiple data sets**, SYSUT2 should be the name of its first data set. DSN1COPY allocates all target data sets. However, you must previously define the target data sets by using IDCAMS.

Adding additional volumes for SYSUT2

When you create a table space or index space by using STOGROUP, the ICF catalog entry has only one volume in the volume list. If the SYSUT2 data set that DSN1COPY restores requires more than one volume, use the IDCAMS command, ALTER ADDVOLUMES, to add additional volume IDs to the integrated catalog entry. The extension to new volumes uses the primary size on each new volume. This is the normal VSAM extension process. If you want the data set to use the secondary size on the candidate volumes, follow these steps:

1. Run DSN1COPY.
2. Run REORG, or make a full image copy and recover the table space.

Performing these steps resets the data set and causes normal extensions through DB2.

Restrictions

This section contains restrictions that you should know about when running DSN1COPY.

DSN1COPY does not alter data set structure. For example, DSN1COPY does not copy a partitioned or segmented table space into a simple table space. The output data set is a page-for-page copy of the input data set. If the intended use of DSN1COPY is to move or restore data, ensure that definitions for the source and target table spaces, tables, and indexes are identical. Otherwise, unpredictable results can occur.

DSN1COPY cannot copy DB2 recovery log data sets. The format of a DB2 log page is different from that of a table or index page. If you try to use DSN1COPY to recover log data sets, DSN1COPY will abnormally terminate.

DSN1COPY will issue an error and terminate if it can be clearly verified that the LOB option is specified, but the data set is not a LOB data set, or that the LOB option is omitted for a data set that is a LOB data set. To avoid problems, always specify the LOB option if the input data set SYSUT1 is a LOB table space, and make sure that the LOB option is not specified for non LOB table spaces.

DSN1COPY cannot copy a source object of 4 GB or greater in size when it is full unless the target object is EA-enabled. For example, the source is full when it is not the last piece of a multi-piece non-partitioned object with a DSSIZE of 4 GB or greater. To avoid VSAM errors and limit each piece to 2 GB so that the target object has more pieces than the original source:

- Define the target data set as EA-enabled and DSN1COPY can be used, one piece at a time, to copy the data from the source that is not EA-enabled to the target.
- If it is not possible to define the target data set as EA-enabled:
  1. Take a full image copy of the entire source object by running the COPY utility and specifying DSNUM ALL.
  2. Allocate the target object by specifying DSSIZE 2G for the DSN1COPY utility.
  3. Define the partition number data sets (2 GB each) with the IDCAMS command. Define enough pieces to hold the entire source.

4. Run the DSN1COPY utility with the image copy as the source (SYSUT1), the target object as SYSUT2, and specify DSSIZE 2G.

Recommendations

This section contains recommendations that you should know about when running the DSN1COPY utility.

Printing with DSN1PRNT instead of DSN1COPY

If you require only a printed hexadecimal dump of a data set, use DSN1PRNT rather than DSN1COPY. For more information, see "Printing with DSN1PRNT instead of DSN1COPY" on page 775.

Determining page size and DSSIZE

Before using DSN1COPY, ensure that you know the page size and data set size (DSSIZE) for the page set. Use the following query on the DB2 catalog to get the information you need, in this example for table 'DEPT':

```
SELECT T.CREATOR,T.NAME,S.NAME AS TABLESPACE,S.PARTITIONS,S.PGSIZE,
      CASE S.DSSIZE
      WHEN 0 THEN
                  CASE WHEN S.TYPE = 'O' THEN 4194304
                  ELSE
                       CASE WHEN S.PARTITIONS > 254 THEN
                            CASE WHEN S.PGSIZE = 4  THEN  4194304
                                 WHEN S.PGSIZE = 8  THEN  8388608
                                 WHEN S.PGSIZE = 16 THEN 16777216
                                 WHEN S.PGSIZE = 32 THEN 33554432
                            ELSE NULL
                            END
                            WHEN S.PARTITIONS > 64 THEN 4194304
                            WHEN S.PARTITIONS > 32 THEN 1048576
                            WHEN S.PARTITIONS > 16 THEN 2097152
                            WHEN S.PARTITIONS > 0  THEN 4194304
                       ELSE 2097152
                       END
                  END
      ELSE S.DSSIZE
      END
      AS DSSIZE
      FROM SYSIBM.SYSTABLES T,
           SYSIBM.SYSTABLESPACE S
      WHERE
           T.NAME = 'DEPT' AND
           T.TSNAME = S.NAME;
```

Figure 134. Example catalog query that returns the page set size and data set size for the page set.

Using the OBIDXLAT option with DSN1COPY

When you use DSN1COPY with the OBIDXLAT option to move objects from one system to another system, ensure that the version information on the target system matches the version information on the source version. For instructions on how to move the objects while ensuring that the version information matches, see "Updating version information when moving objects to another subsystem" on page 518.

For more information about versions and how DB2 uses them, see Part 2 of *DB2 Administration Guide*.

Using DSN1COPY to copy data sets

This section describes the following tasks that are associated with running the DSN1COPY utility:

The effect of altering a table before running DSN1COPY

When you use ALTER TABLE ADD COLUMN, the table does not change; only the description of the table changes. You must run REORG on the table space (so that the data matches its description) before you can run DSN1COPY on the table space.

Checking for inconsistent data

When critical data is involved, use the CHECK option to prevent the undetected copying of inconsistent data to the output data set. The CHECK option of DSN1COPY performs validity checking on one page at a time.

You must run a CHECK utility job on the table space that is involved to ensure that no inconsistencies exist between data and indexes on that data:

- Before using DSN1COPY to save critical data that is indexed
- After using DSN1COPY to restore critical data that is indexed

The CHECK utility performs validity checking between pages.

The effects of not specifying the OBIDXLAT option

If you use DSN1COPY to load data into a table space or index without specifying the OBIDXLAT option, be careful not to invalidate embedded DB2 internal identifiers. Those OBIDs can become invalid in the following circumstances:

- When you drop and re-create tables after the input data set to DSN1COPY was created.
- When a difference exists among the following attributes between the target subsystem and the source subsystem:
  - Table space attributes of BUFFERPOOL or NUMPARTS
  - Table attributes other than table name, table space name, and database name
  - The order of the table spaces, indexes, and tables that the user defined or dropped in the source and target databases

To protect against invalidating the OBIDs, specify the OBIDXLAT parameter for DSN1COPY. The OBIDXLAT parameter translates OBID, DBID, or PSID before DSN1COPY copies the data.

Requirements for using an image copy as input to DSN1COPY

If you want to use image copies as input to DSN1COPY, you must produce those image copies by using the COPY utility with SHRLEVEL REFERENCE. Using the

FULLCOPY parameter ensures that the data that is contained in your image copies is consistent. DSN1COPY accepts an index image copy as input when you specify the FULLCOPY option.

Resetting page log RBAs

Use the RESET option to reset the log RBAs that are recorded in a table space or index space and the high-formatted page number in the header page to 0. DSN1COPY performs CHECK processing, regardless of whether you explicitly requested CHECK.

Do not specify the RESET parameter for page sets that are in group buffer pool RECOVER-pending (GRECP) status.

Copying from an image copy

When you use DSN1COPY to copy from an image copy of a table space's data sets to the table space's data sets, specify the following SYSUT2 data sets:

- **If SYSUT1 is an image copy of a single partition**, ensure that the first data set of the table space is named SYSUT2. DSN1COPY determines the correct target data set. Code the NUMPARTS(*nn*) parameter, where *nn* is the number of partitions in the entire table space. However, if the partitioned table space is defined with more than one VCAT name (for example, a unique VCAT for different partitions), use SYSUT2 as the name of the data set for that partition.
- **If SYSUT1 is an image copy of an entire partitioned table space**, ensure that the first data set of the table space is named SYSUT2. In this case, DSN1COPY allocates all of the target data sets. However, you must have previously defined the target data sets by using Access Method Services. Code the NUMPARTS parameter as described in the first bullet when the table space is partitioned. When multiple VCAT names are used for different partitions of a partitioned table space, DSN1COPY cannot restore the entire table space by using as input a single full image copy of the table space. In this case, when you use DSN1COPY, you must restore individual copies of each partition by using the name of the data sets for that partition. Code the NUMPARTS(*nn*) parameter, where *nn* is the number of partitions in the entire table space.
- **If SYSUT1 is an image copy of a single data set of a multiple data set linear table space,** ensure that the actual (not the first) output data set is named SYSUT2. Do not specify NUMPARTS because this parameter is only for partitioned table spaces.
- **If SYSUT1 is an image copy of an entire multiple data set linear table space,** ensure that the first data set of the table space is named SYSUT2. DSN1COPY allocates all target data sets.

Restoring indexes with DSN1COPY

When a table space is restored using either the TOCOPY option of RECOVER or the DSN1COPY utility, restore the indexes in one of the following three ways:

- Use the RECOVER utility, if you have a full image copy available, and the index was defined with the COPY YES option.
- Use DSN1COPY on the indexes, if a copy is available. If you specified the OBIDXLAT option for the data, you must also specify the OBIDXLAT option for the indexes. Also, the indexes must all have been copied at the same time as the data; otherwise, inconsistencies might exist.
- If you do not have an image copy of the index, use the REBUILD INDEX utility, which reconstructs the indexes from the data. For more information about the REBUILD INDEX utility, refer to Chapter 22, "REBUILD INDEX," on page 335.

Restoring table spaces with DSN1COPY

You cannot use RECOVER TOCOPY for an image-copy data set that is not referenced by SYSIBM.SYSCOPY for that table space or data set. An attempt to do so results in the message "TOCOPY DATASET NOT FOUND".

The MODIFY utility might have removed the row in SYSIBM.SYSCOPY. If the row has been removed, and if the image copy is a full image copy with SHRLEVEL REFERENCE, use DSN1COPY to restore the table space or data set.

DSN1COPY can restore the object to an incremental image copy, but it must first restore the previous full image copy and any intermediate incremental image copies. These actions ensure data integrity. You are responsible for providing the correct sequence of image copies. DB2 cannot help ensure the proper sequence.

If you use DSN1COPY for point-in-time recovery, the table space is not recoverable with the RECOVER utility. Because DSN1COPY executed outside of DB2's control, DB2 is not aware that you recovered to a point in time. Use DSN1COPY to recover the affected table space after point-in-time recovery. Then perform the following steps:

1. Remove old image copies by using MODIFY AGE.
2. Create one or more full image copies by using SHRLEVEL REFERENCE.

Printing with DSN1COPY

If you want to print one or more pages without invoking the utility's copy function, use DSN1PRNT to avoid unnecessary reading of the input file.

When you use DSN1COPY for printing, you must specify the PRINT parameter. The requested operation takes place only for the specified data set. If the input data set belongs to a linear table space or index space that is larger than 2 GB, specify the correct data set. Alternatively, if it is a partitioned table space or partitioned index, specify the correct data set. For example, DSN1COPY prints a page range in the second partition of a four-partition table space. DSN1COPY does this by specifying NUMPARTS(4) and the data set name of the second data set in the VSAM group (DSN=...A002).

To print a full image copy data set (rather than recovering a table space), specify a DUMMY SYSUT2 DD statement, and specify the FULLCOPY parameter.

Copying tables from one subsystem to another

When you copy tables from one subsystem to another, you must ensure that the version information on the target subsystem matches the version information on the source subsystem. For instructions on how to ensure that this information matches, see "Updating version information when moving objects to another subsystem" on page 518.

Be careful when you copy a table that contains an identity column from one DB2 subsystem to another:

1. Stop the table space on the source subsystem.
2. Issue a SELECT statement to query the SYSIBM.SYSSEQUENCES entry that corresponds to the identity column for this table on the source subsystem. Add the INCREMENT value to the MAXASSIGNEDVAL to determine the next value (*nv*) for the identity column.

3. Create the table on the target subsystem. On the identity column specification, specify *nv* for the START WITH value, and ensure that all of the other identity column attributes are the same as for the source table.
4. Stop the table space on the target subsystem.
5. Copy the data by using DSN1COPY.
6. Start the table space on the source subsystem for read-write access.
7. Start the table space on the target subsystem for read-write access.

Sample DSN1COPY control statements

If you run online REORG with the FASTSWITCH option, the fifth-level qualifier in the data set name can be either I0001 or J0001. These examples use I0001.

Example 1: Checking input data set before copying. The following statement specifies that the DSN1COPY utility is to copy the data set that is identified by the SYSUT1 DD statement to the data set that is identified by the SYSUT2 DD statement. Before DSN1COPY copies this data, the utility is to check the validity of the input data set.

```
//RUNCOPY EXEC PGM=DSN1COPY,PARM='CHECK'
//* COPY VSAM TO SEQUENTIAL AND CHECK PAGES
//STEPLIB  DD DSN=PDS CONTAINING DSN1COPY
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSN=DSNCAT.DSNDBC.DSNDB01.SYSUTILX.I0001.A001,DISP=OLD
//SYSUT2   DD DSN=TAPE.DS,UNIT=TAPE,DISP=(NEW,KEEP),VOL=SER=UTLBAK
```

Example 2: Translating the DB2 internal identifiers. The statement in Figure 135 specifies that DSN1COPY is to copy the data set that is identified by the SYSUT1 DD statement to the data set that is identified by the SYSUT2 DD statement. The OBIDXLAT option specifies that DSN1COPY is to translate the OBIDs before the data set is copied. The OBIDs are provided as input on the SYSXLAT DD statement. Because the OBIDXLAT option is specified, DSN1COPY also checks the validity of the input data set, even though the CHECK option is not specified.

```
//EXECUTE EXEC PGM=DSN1COPY,PARM='OBIDXLAT'
//STEPLIB  DD  DSN=PDS CONTAINING DSN1COPY
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  DSN=DSNC810.DSNDBC.DSN8D81P.DSN8S81C.I0001.A001,
//             DISP=OLD
//SYSUT2   DD  DSN=DSNC618.DSNDBC.DSN8D81P.DSN8S81C.I0001.A001,
//             DISP=OLD
//SYSXLAT  DD  *
260,280
2,10
3,55
6,56
7,57
/*
```

Figure 135. Example DSN1COPY statement with the OBIDXLAT option.

Example 3: Printing a single page of a partitioned table space. The following statement specifies that DSN1COPY is to print page 2002A1 of the table space in the data set that is identified by the SYSUT1 DD statement. This table space has eight partitions, as indicated by the NUMPARTS option.

```
//PRINT EXEC PGM=DSN1COPY,PARM='PRINT(2002A1),NUMPARTS(8)'
//* PRINT A PAGE IN THE THIRD PARTITION OF A TABLE SPACE CONSISTING
//* OF 8 PARTITIONS.
```

```
//SYSUDUMP DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSUT2   DD DUMMY
//SYSUT1   DD DSN=DSNCAT.DSNDBD.MMRDB.PARTEMP1.I0001.A003,DISP=OLD
```

Example 4: Printing 16 pages of a nonpartitioning index. The following statement specifies that DSN1COPY is to print 16 pages of a nonpartitioning index in the data set that is identified by the SYSUT1 DD statement. The pages range from page F0000 to page F000F, as indicated by the PRINT option. The maximum data set size is 64 MB, as indicated by the PIECESIZ option.

```
//PRINT2 EXEC PGM=DSN1COPY,PARM=(PRINT(F0000,F000F),PIECESIZ(64M))
//* PRINT THE FIRST 16 PAGES IN THE 61ST PIECE OF AN NPI WITH PIECE SIZE OF 64M
//SYSUDUMP DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSUT2    DD DUMMY
//SYSUT1    DD DISP=OLD,DSN=DSNCAT.DSTDBD.MMRDB.NPI1.I0001.A061
```

Example 5: Copying individual partitions of a partitioned table space. In the example in Figure 136, the two job steps specify that DSN1COPY is to copy partitions 1501 and partition 1502 from image copy data sets into a partitioned table space. In the two SYSUT2 DD statements, the fifth-level qualifier in the data set names can differ, because each job step lists an individual partition. The FULLCOPY option is used in both steps to indicate that the input data set is a full image copy. The NUMPARTS option indicates that the input data set has 1600 partitions. The RESET option resets to 0 the high-formatted page number in the header page. Because this option is specified, DSN1COPY checks the validity of the input data, even though the CHECK option is not specified.

```
//STEP1    EXEC PGM=DSN1COPY,
//            PARM='NUMPARTS(1600),RESET,FULLCOPY'
//SYSUDUMP DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DISP=SHR,DSN=PROD.IMAGE.COPY.PART1501
//SYSUT2   DD DISP=OLD,DSN=DSNCAT.DSNDBD.TESTDB.TS01.I0001.B501
//STEP2    EXEC PGM=DSN1COPY,
//            PARM='NUMPARTS(1600),RESET,FULLCOPY'
//SYSUDUMP DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DISP=SHR,DSN=PROD.IMAGE.COPY.PART1502
//SYSUT2   DD DISP=OLD,DSN=DSNCAT.DSNDBD.TESTDB.TS01.J0001.B502
```

Figure 136. Example DSN1COPY job for partitions

Example 6: Copying all partitions of a partitioned table space. The following statement specifies that DSN1COPY is to copy data into all partitions of a partitioned table space by using a full image copy of the table space as input. The input image copy has 16 partitions, as indicated by the NUMPARTS option. You must ensure that the fifth-level qualifier in the data set name is the same, either I0001 or J0001, for **all** partitions of the output table space before running this type of job stream.

```
//DSN1COPY EXEC PGM=DSN1COPY,
//            PARM='NUMPARTS(16),RESET,FULLCOPY'
//SYSUDUMP DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DISP=SHR,DSN=PROD.IMAGE.COPY.DSNUMALL
//SYSUT2   DD DISP=OLD,DSN=DSNCAT.DSNDBD.TESTDB.TS01.I0001.A001
```

DSN1COPY output

One intended use of this utility is to aid in determining and correcting system problems. When diagnosing DB2, you might need to refer to licensed documentation to interpret output from this utility. For more information about diagnosing problems, see *DB2 Diagnosis Guide and Reference*.

Chapter 41. DSN1LOGP

The DSN1LOGP utility formats the contents of the recovery log for display. The two recovery log report formats are:

- A detail report of individual log records. This information helps IBM Software Support personnel analyze the log in detail. (This book does not include a full description of the detail report.)
- A summary report, which helps you:
  - Perform a conditional restart
  - Resolve indoubt threads with a remote site
  - Detect problems with data propagation

You can specify the range of the log to process and select criteria within the range to limit the records in the detail report. For example, you can specify:

- One or more units of recovery that are identified by URID
- A single database

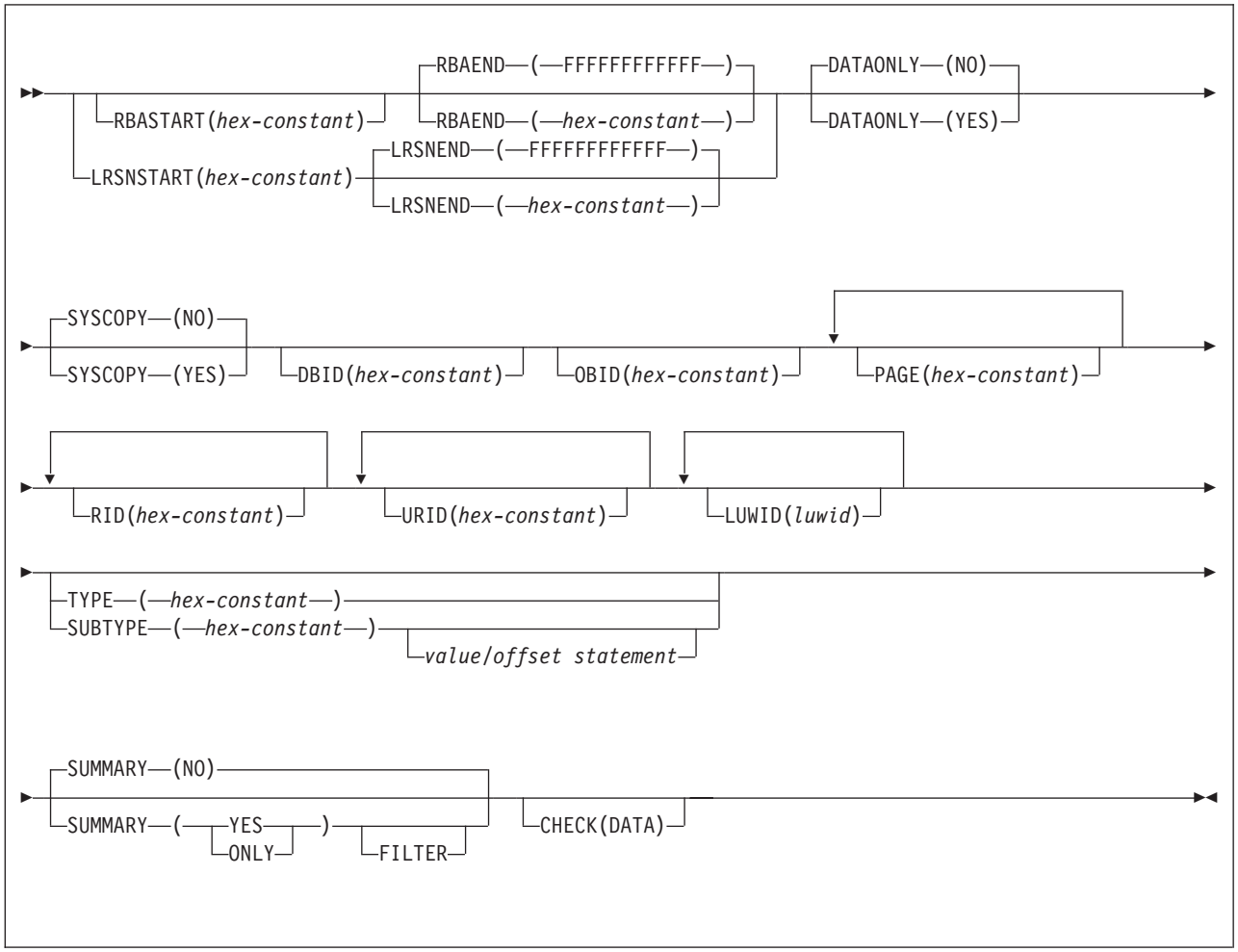
By specifying a URID and a database, you can display recovery log records that correspond to the use of one database by a single unit of recovery.

DSN1LOGP cannot read logs that have been compressed by DFSMS. (This compression requires extended format data sets.)

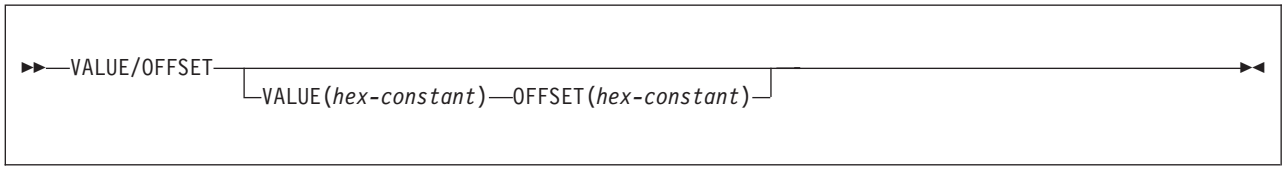
The following topics provide additional information:

Syntax and options of the DSN1LOGP control statement

DSN1LOGP syntax diagram



value/offset statement:



Option descriptions

To execute DSN1LOGP, construct a batch job. The utility name, DSN1LOGP, should appear on the EXEC statement, as shown in “Sample DSN1LOGP control statements” on page 757.

Specify keywords in up to 50 control statements in the SYSIN file. Each control statement can have up to 72 characters. To specify no keywords, either use a SYSIN file with no keywords following it, or omit the SYSIN file from the job JCL.

If you specify more than one keyword, separate them by commas. You can specify the keywords in any order. You can include blanks between keywords, and also between the keywords and the corresponding values.

RBASTART(*hex-constant***)**
Specifies the hexadecimal log RBA from which to begin reading. If the value does not match the beginning RBA of one of the log records, DSN1LOGP begins reading at the beginning RBA of the next record. For any given job, specify this keyword only once. Alternative spellings: STARTRBA, ST.

hex-constant is a hexadecimal value consisting of 1 to 12 characters (6 bytes); leading zeros are not required.

The **default** is **0**.

RBAEND(*hex-constant***)**
Specifies the last valid hexadecimal log RBA to extract. If the specified RBA is in the middle of a log record, DSN1LOGP continues reading the log in an attempt to return a complete log record.

To read to the last valid RBA in the log, specify RBAEND(FFFFFFFFFFFF). For any given job, specify this keyword only once. Alternative spellings: ENDRBA, EN.

hex-constant is a hexadecimal value consisting of 1 to 12 characters (6 bytes); leading zeros are not required.

The **default** is **FFFFFFFFFFFF**.

RBAEND can be specified only if RBASTART is specified.

LRSNSTART(*hex-constant***)**
Specifies the log record sequence number (LRSN) from which to begin the log scan. DSN1LOGP starts its processing on the first log record that contains an LRSN value that is greater than or equal to the LRSN value that is specified on LRSNSTART. The default LRSN is the LRSN at the beginning of the data set. Alternative spellings: STARTLRSN, STRTLRSN, and LRSNSTRT.

For any given job, specify this keyword only once.

You must specify this keyword to search the member BSDSs and to locate the log data sets from more than one DB2 subsystem. You can specify either the LRSNSTART keyword or the RBASTART keyword to search the BSDS of a single DB2 subsystem and to locate the log data sets.

LRSNEND(*hex-constant***)**
Specifies the LRSN value of the last log record that is to be scanned. When LRSNSTART is specified, the default is X'FFFFFFFFFFFF'. Otherwise, it is the end of the data set. Alternative spelling: ENDLRSN.

For any given job, specify this keyword only once.

DATAONLY Limits the log records in the detail report to those that represent data changes (insert, page repair, update space map, and so on).

The **default** is **DATAONLY**(**NO**).

(YES) Extracts log records for data changes only. For example,

DATAONLY(YES), together with a DBID and OBID, reads only the log records that modified data for that DBID and OBID.

(NO) Extracts all record types.

SYSCOPY Limits the detail report to SYSCOPY log records. The **default** is **SYSCOPY**(**NO**).

(YES) Includes only SYSCOPY log records in the detail report.

(NO) Does not limit records to SYSCOPY records only.

DBID(*hex-constant***)**

Specifies a hexadecimal database identifier (DBID). DSN1LOGP extracts only the records that are associated with that DBID. For any given job, specify this keyword only once.

hex-constant is a hexadecimal value consisting of one to four characters. Leading zeros are not required.

The DBID is displayed in many DB2 messages. You can also find the DBID in the DB2 catalog for a specific object (for example, in the column named DBID of the SYSIBM.SYSTABLESPACE catalog table).

When you select a DBID from a catalog table, the value is displayed in decimal format. Use the SQL HEX function in a SELECT statement to convert a DBID to hexadecimal format. The following SQL statements show this use of the HEX function:

```
SELECT NAME, DBNAME, HEX(DBID), HEX(PSID)
FROM SYSIBM.SYSTABLESPACE
WHERE NAME ='table space name'

SELECT NAME, DBNAME, HEX(DBID), HEX(ISOBID)
FROM SYSIBM.SYSINDEXES
WHERE NAME ='index name'
```

OBID(*hex-constant***)**

Specifies a hexadecimal database object identifier, either a data page set identifier (PSID) or an index page set identifier (ISOBID). DSN1LOGP extracts only the records that are associated with that identifier.

hex-constant is a hexadecimal value consisting of one to four characters. Leading zeros are not required.

Whenever DB2 makes a change to data, the log record that describes the change identifies the database by DBID and the table space by page set ID (PSID). You can find the PSID column in the SYSIBM.SYSTABLESPACE catalog table.

You can also find a column named OBID in the SYSIBM.SYSTABLESPACE catalog table. That column actually contains the OBID of a file descriptor; don't confuse this with the PSID, which is the information that you must include when you execute DSN1LOGP.

Whenever DB2 makes a change to an index, the log record that describes the change identifies the database (by DBID) and the index space (by index space OBID or ISOBID). You can find the ISOBID for an index space in the column named ISOBID in the SYSIBM.SYSINDEXES catalog table.

You can also find a column named OBID in the SYSIBM.SYSINDEXES catalog table. This column actually contains the identifier of a fan set descriptor; don't confuse this with the ISOBID, which is the information that you must include when you execute DSN1LOGP.

When you select either the PSID or the ISOBID from a catalog table, the value is displayed in decimal format. Use the SQL HEX function in your select statement to convert them to hexadecimal.

For any given DSN1LOGP job, use this keyword only once. If you specify OBID, you must also specify DBID.

PAGE(*hex-constant***)**

Specifies a hexadecimal page number. When data or an index is changed, a recovery log record is written to the log, identifying the object identifier and the page number of the changed data page or index page. Specifying a page number limits the search to a single page; otherwise, all pages for a given combination of DBID and OBID are extracted. The log output also contains page set control log records for the specified DBID and OBID, and system event log records, unless DATAONLY(YES) is also specified.

hex-constant is a hexadecimal value consisting of a maximum of eight characters.

You can specify a maximum of 100 PAGE keywords in any given DSN1LOGP job. You must also specify the DBID and OBID keywords that correspond to those pages.

The PAGE and RID keywords are mutually exclusive.

RID(*hex-constant***)**

Specifies a record identifier, which is a hexadecimal value consisting of 10 characters, with the first eight characters representing the page number and the last two characters representing the page ID map entry number. The option limits the log records that are extracted to those that are associated with that particular record. The log records that are extracted include not only those that are directly associated with the RID, such as insert and delete, but also the control records that are associated with the DBID and OBID specifications, such as page set open, page set close, set write, reset write, page set write, data set open, and data set close.

You can specify a maximum of 40 RID keywords in any given DSN1LOGP job. You must also specify the DBID and OBID keywords that correspond to the specified records.

The PAGE and RID keywords are mutually exclusive.

URID(*hex-constant***)**

Specifies a hexadecimal unit of recovery identifier (URID). Changes to data and indexes occur in the context of a DB2 unit of recovery, which is identified on the log by a BEGIN UR record. In the summary DSN1LOGP report, the URID is listed in the STARTRBA field in message DSN1162I. In the detail DSN1LOGP report, look for the subtype of BEGIN UR; the URID is listed in the URID field. Using the log RBA of that record as the URID value limits the extraction of information from the DB2 log to that unit of recovery.

hex-constant is a hexadecimal value consisting of 1 to 12 characters (6 bytes). Leading zeros are not required.

You can specify a maximum of 10 URID keywords in any given DSN1LOGP job.

LUWID(*luwid***)** Specifies up to 10 LUWIDs that DSN1LOGP is to include information about in the summary report.

luwid consists of three parts: an LU network name, an LUW instance number, and a commit sequence number. If you supply the first two parts, the summary report includes an entry for each commit that is performed in the logical unit of work (within the search range). If you supply all three parts, the summary report includes an entry for only that LUWID.

The LU network name consists of a one- to eight-character network ID, a period, and a one- to eight-character network LU name. The LUW instance number consists of a period, followed by 12 hexadecimal characters. The last element of the LUWID is the commit sequence number of 4 hexadecimal characters, preceded by a period.

TYPE(*hex-constant***)**

Limits the log records that are extracted to records of a specified type. The TYPE and SUBTYPE options are mutually exclusive.

hex-constant indicates the type, as follows:

Constant	Description
2	Page set control record
4	SYSCOPY utility record
10	System event record
20	UR control record
100	Checkpoint record
200	UR-UNDO record
400	UR-REDO record
800	Archive quiesce record
1000 to 8000	Assigned by the resource manager

SUBTYPE(*hex-constant***)**

Restricts formatting to a particular subtype of unit of recovery undo and redo log records (types 200 and 400). The TYPE and SUBTYPE options are mutually exclusive.

hex-constant indicates the subtype, as follows:

Constant	Description
1	Update data page
2	Format page or update space map
3	Update space map bits
4	Update to index space map
5	Update to index page

6 DBA table update log record

7 Checkpoint DBA table log record

9 DBD virtual memory copy

A Exclusive lock on page set partition or DBD

B Format file page set

C Format index page set

F Update by repair (first half if 32 KB)

10 Update by repair (second half if 32 KB)

11 Allocate or deallocate a segment entry

12 Undo/redo log record for modified page or redo log record for formatted page

14 Savepoint

15 Other DB2 component log records that are written for RMID 14

17 Checkpoint record of modified page set

19 Type 2 index update

1A Type 2 index undo/redo or redo log record

1B Type 2 index change notification log record

1C Type 2 index space map update

1D DBET log record with exception data

1E DBET log record with LPL/GRECP data

65 Data propagation diagnostic log

81 Index dummy compensation log record

82 START DATABASE ACCESS (FORCE) log record

The VALUE and OFFSET options must be used together. You can specify a maximum of 10 VALUE-OFFSET pairs. The SUBTYPE parameter is required when using the VALUE and OFFSET options.

VALUE(*hex-constant***)**
Specifies a value that must appear in a log record that is to be extracted.

hex-constant is a hexadecimal value consisting of a maximum of 64 characters and must be an even number of characters.

The SUBTYPE keyword must be specified before the VALUE option.

OFFSET(*hex-constant***)**
Specifies an offset from the log record header at which the value that is specified in the VALUE option must appear.

hex-constant is a hexadecimal value consisting of a maximum of eight characters.

The SUBTYPE keyword must be specified before specifying the OFFSET option.

SUMMARY Summarizes all recovery information within the RBASTART and RBAEND specifications. You can use summary information to determine what work is incomplete when DB2 starts. You cannot limit the output of the summary report with any of the other options, except by using the FILTER option with a URID or LUWID specification. The **default** is **SUMMARY**(**NO**).

(YES) Generates both a detail and summary report.

(NO) Generates only a detail report.

(ONLY)
Generates only a summary report.

FILTER Restricts the summary report to include messages for only the specified URIDs and LUWIDs. Specify this option only once.

The SUMMARY keyword must be specified before FILTER.

CHECK(DATA)
Specifies that DSN1LOGP is to check the specified range of data pages for page regression. Any page regression errors are displayed in the detail and summary reports. See "Description of the report on page regression errors" on page 764 for a sample report on page regression errors.

Before running DSN1LOGP

This section contains information that you need to know before running DSN1LOGP.

Environment

DSN1LOGP runs as a batch z/OS job.

DSN1LOGP runs on archive data sets, but not active data sets, when DB2 is running.

Authorization required

DSN1COPY does not require authorization. However, if any of the data sets is RACF-protected, the authorization ID of the job must have RACF authority.

Control statement

Create the utility control statement for the DSN1LOGP job. See "Syntax and options of the DSN1LOGP control statement" on page 748 for DSN1LOGP syntax and option descriptions.

Required data sets

When you execute DSN1LOGP, provide the following data definition (DD) statements:

SYSPRINT DSN1LOGP writes all error messages, exception conditions, and the detail report to the SYSPRINT file. The logical record length (LRECL) is 131.

SYSIN DSN1LOGP specifies keywords in this file. The LRECL must be 80. Keywords and values must appear in characters 1 through 72.

DSN1LOGP allows specification of as many as 50 control statements for a given job. DSN1LOGP concatenates all records into a single string.

SYSSUMRY DSN1LOGP writes the formatted output of a summary report to the SYSSUMRY file. The LRECL is 131. For an example of the appropriate JCL, see "Example 4: Use DSN1LOGP with the SUMMARY option" on page 758.

DSN1LOGP identifies the recovery log by DD statements that are described in the stand-alone log services. For a description of these services, see Appendix C (Volume 2) of *DB2 Administration Guide*.

Identifying log data sets

You must identify to DSN1LOGP the log data sets that are to be processed by including at least one of the following DD statements.

BSDS The BSDS identifies and provides information about all active log data sets and archive log data sets that exist in your DB2 subsystem. When you identify the BSDS to DSN1LOGP, you must provide the beginning and ending RBAs for the range of the recovery log that you want displayed. DSN1LOGP then associates the beginning RBA specifications and the ending RBA specifications with the appropriate data set names.

See "Example 1: Extracting information from the recovery log with an available BSDS." on page 757 for guidance in using this DD statement.

ACTIVE*n* If the BSDS is not available, and if the active log data sets that are involved have been copied and sent to you, you can specify the set of active log data sets that are to be processed by DSN1LOGP by specifying one or more ACTIVE DD statements. If you used the REPRO command of Access Method Services for copying the active log, you must identify this data set in an ARCHIVE DD statement.

Each DD statement that you include identifies another active log data set. If you identify more than one active log data set, you must list the ACTIVE*n* DD statements in ascending log RBA sequence. For example, ACTIVE1 must identify a portion of the log that is less than ACTIVE2, and ACTIVE2 must identify a portion of the log that is less than ACTIVE3. If you do not specify this correctly, errors that DSN1LOGP does not detect can occur.

When you identify active log data sets, you do not need to use the RBASTART and RBAEND keywords (as you do when you identify the BSDS). DSN1LOGP scans all active log data sets that the job indicates only when the data sets are in the correct log RBA sequence.

See "Example 2: Extracting information from the active log when the BSDS is not available" on page 758 for guidance in using these DD statements.

ARCHIVE If the BSDS is not available (as previously described under ACTIVE*n*), you can specify which archive log data sets are to be processed by specifying one ARCHIVE DD statement, concatenated with one or more DD statements.

Each DD statement that you include identifies another archive log data set. If you identify more than one archive log data set, you must list the DD statements that correspond to the multiple archive

log data sets in ascending log RBA sequence. If you do not specify this correctly, errors that DSN1LOGP does not detect can occur.

When you identify archive log data sets, you do not need to use the RBASTART and RBAEND keywords. DSN1LOGP scans all archive log data sets that are indicated by the job only when the data sets are in the correct log RBA sequence.

See "Example 3: Extracting information from the archive log when the BSDS is not available" on page 758 for guidance in using the ARCHIVE DD statement.

Data sharing requirements: When selecting log records from more than one DB2 subsystem, you must use all of the following DD statements to locate the log data sets:

GROUP
M*xx*BSDS
M*xx*ARCHV
M*xx*ACT*n*

See Appendix C (Volume 2) of *DB2 Administration Guide* for descriptions of those statements. If you use GROUP or M*xx*BSDSs to locate the log data sets, you must use LRSNSTART to define the selection range.

Using DSN1LOGP to format the contents of the recovery log

This section describes the following tasks that are associated with running the DSN1LOGP utility:

Reading archive log data sets on tape

If you store your archive logs on tape, DSN1LOGP constructs two files on tape during the archiving process. The first file is the BSDS, and the second is a dump of the active log that DSN1LOGP is currently archiving. If a failure occurs during the time DSN1LOGP is archiving the BSDS, DB2 might omit the BSDS. In this case, the first file contains the active log.

If you perform archiving on tape, the first letter of the lowest-level qualifier varies for both the first and second data sets. The first letter of the first data set is B (for BSDS), and the first letter of the second data set is A (for archive). Hence, the archive log data set names all end in A*xxxxxx*, and the DD statement identifies each of them as the second data set on the corresponding tape:

```
LABEL=(2,SL)
```

When reading archive log data sets on tape (or copies of active log data sets on tape), add one or more of the following Job Entry Subsystem (JES) statements:

For the JES3 environment:

JES3 environment JCL	Description
//*MAIN SETUP=JOB	Alert the z/OS operator to mount the initial volumes before the job executes.
//*MAIN HOLD=YES	Place the job in HOLD status until the operator is ready to release the job.

TYPRUN=HOLD	Perform the same function as //*MAIN HOLD=YES. The system places the JCL on the JOB statement.

For the JES2 environment:

JES2 environment JCL	Description
/*SETUP	Alert the z/OS operator to prepare to mount a specified list of tapes.
/*HOLD	Place the job in HOLD status until the operator has located the tapes and is ready to release the job.
TYPRUN=HOLD	Perform the same function as /*HOLD. The system places the JCL on the JOB statement.

Alternatively, submit the job to a z/OS initiator that your operations center has established for exclusive use by jobs that require tape mounts. Specify the initiator class by using the CLASS parameter on the JOB statement, in both JES2 and JES3 environments.

For additional information on these options, refer to *z/OS MVS JCL User's Guide* or *z/OS MVS JCL Reference.*

Locating table and index identifiers

Use the DSN1PRNT utility to find the DBIDs, PSIDs, ISOBIDs, and OBIDs of the tables and indexes from the system tables. For more information, see Chapter 42, "DSN1PRNT," on page 767.

Sample DSN1LOGP control statements

Example 1: Extracting information from the recovery log with an available BSDS. The following example shows how to extract information from the recovery log when you have the BSDS available. The extraction starts at the log RBA of X'AF000' and ends at the log RBA of X'B3000'. The DSN1LOGP utility identifies the table or index space by the DBID of X'10A' (266 decimal) and the OBID of X'1F' (31 decimal).

```
//STEP1 EXEC PGM=DSN1LOGP
//STEPLIB  DD DSN=PDS containing DSN1LOGP
//SYSPRINT DD SYSOUT=A
//SYSABEND DD SYSOUT=A
//BSDS     DD DSN=DSNCAT.BSDS01,DISP=SHR
//SYSIN    DD *
  RBASTART (AF000) RBAEND (B3000)
  DBID (10A) OBID(1F)
/*
```

You can think of the DB2 recovery log as a large sequential file. When recovery log records are written, they are written to the end of the log. A log RBA is the address of a byte on the log. Because the recovery log is larger than a single data set, the recovery log is physically stored on many data sets. DB2 records the RBA ranges and their corresponding data sets in the BSDS. To determine which data set contains a specific RBA, read the information about the DSNJU004 utility under Chapter 37, "DSNJU004 (print log map)," on page 697 and see Part 4 (Volume 1) of *DB2 Administration Guide*. During normal DB2 operation, messages are issued that include information about log RBAs.

Example 2: Extracting information from the active log when the BSDS is not available. The following example shows how to extract the information from the active log when the BSDS is not available. The extraction includes log records that apply to the table space or index space that is identified by the DBID of X'10A' and the OBID of X'1F'. The only information that is extracted is information that relates to page numbers X'3B' and X'8C', as identified by the PAGE options. You can omit beginning and ending RBA values for ACTIVE*n* or ARCHIVE DD statements because the DSN1LOGP search includes all specified ACTIVE*n* DD statements. The DD statements ACTIVE1, ACTIVE2, and ACTIVE3 specify the log data sets in ascending log RBA range. Use the DSNJU004 utility to determine what the log RBA range is for each active log data set. If the BSDS is not available and you cannot determine the ascending log RBA order of the data sets, you must run each log data set individually.

```
//STEP1 EXEC PGM=DSN1LOGP
//STEPLIB  DD DSN=PDS containing DSN1LOGP
//SYSPRINT DD SYSOUT=A
//SYSABEND DD SYSOUT=A
//ACTIVE1  DD DSN=DSNCAT.LOGCOPY1.DS02,DISP=SHR      RBA X'A000' - X'BFFF'
//ACTIVE2  DD DSN=DSNCAT.LOGCOPY1.DS03,DISP=SHR      RBA X'C000' - X'EFFF'
//ACTIVE3  DD DSN=DSNCAT.LOGCOPY1.DS01,DISP=SHR      RBA X'F000' - X'12FFF'
//SYSIN    DD *
  DBID (10A) OBID(1F) PAGE(3B) PAGE(8C)
/*
```

Example 3: Extracting information from the archive log when the BSDS is not available. The example in Figure 137 shows how to extract the information from archive logs when the BSDS is not available. The extraction includes log records that apply to a single unit of recovery (whose URID is X'61F321'). Because the BEGIN UR is the first record for the unit of recovery and is at X'61F321', the beginning RBA is specified to indicate that it is the first RBA in the range from which to extract recovery log records. Also, because no ending RBA value is specified, all specified archive logs are scanned for qualifying log records. The specification of DBID(4) limits the scan to changes that the specified unit of recovery made to all table spaces and index spaces in the database whose DBID is X'4'.

```
//STEP1 EXEC PGM=DSN1LOGP
//STEPLIB  DD DSN=PDS containing DSN1LOGP
//SYSPRINT DD SYSOUT=A
//SYSABEND DD SYSOUT=A
//ARCHIVE  DD DSN=DSNCAT.ARCHLOG1.A0000037,UNIT=TAPE,VOL=SER=T10067,
//            DISP=(OLD,KEEP),LABEL=(2,SL)
//         DD DSN=DSNCAT.ARCHLOG1.A0000039,UNIT=TAPE,VOL=SER=T30897,
//            DISP=(OLD,KEEP),LABEL=(2,SL)
//         DD DSN=DSNCAT.ARCHLOG1.A0000041,UNIT=TAPE,VOL=SER=T06573,
//            DISP=(OLD,KEEP),LABEL=(2,SL)
//SYSIN    DD *
  RBASTART (61F321)
  URID (61F321) DBID(4)
/*
```

Figure 137. Example DSN1LOGP statement with RBASTART and URID options

Example 4: Use DSN1LOGP with the SUMMARY option. The DSN1LOGP SUMMARY option allows you to scan the recovery log to determine what work is incomplete at restart time. You can specify this option either by itself or when you use DSN1LOGP to produce a detail report of log data. Summary log results appear in SYSSUMRY; therefore, you must include a SYSSUMRY DD statement as part of the JCL with which you execute DSN1LOGP.

The following example produces both a detail and a summary report that uses the BSDS to identify the log data sets. The summary report summarizes all recovery log information within the RBASTART and RBAEND specifications. You cannot limit the output of the summary report with any of the other options, except by using the FILTER option with a URID or LUWID specification. RBASTART and RBAEND specification use depends on whether a BSDS is used.

This example is similar to Example 1, in that it shows how to extract the information from the recovery log when you have the BSDS available. However, this example also shows you how to specify a summary report of all logged information between the log RBA of X'AF000' and the log RBA of X'B3000'. This summary is generated with a detail report, but it is printed to SYSSUMRY separately.

```
//STEP1 EXEC PGM=DSN1LOGP
//STEPLIB  DD DSN=PDS containing DSN1LOGP
//SYSPRINT DD SYSOUT=A
//SYSSUMRY DD SYSOUT=A
//SYSABEND DD SYSOUT=A
//BSDS     DD DSN=DSNCAT.BSDS01,DISP=SHR
//SYSIN    DD *
  RBASTART (AF000) RBAEND (B3000)
  DBID (10A) OBID(1F) SUMMARY(YES)
/*
```

Example 5: Use DSN1LOGP on all members of a data sharing group. The following example shows how to extract log information that pertains to the table space that is identified by DBID X'112' and OBID X'1D' from all members of a data sharing group.

```
//STEP1 EXEC PGM=DSN1LOGP
//STEPLIB DD  DSN=PDS containing DSN1LOGP
//SYSPRINT SYSOUT=A
//SYSABEND SYSOUT=A
//GROUP DD DSN=DSNDB0G.BSDS01,DISP=SHR
//SYSIN DD *
  DATAONLY (YES)
  LRSNSTART (A7951A001AD5) LRSNEND (A7951A003B6A)
  DBID (112) OBID(1D)
/*
```

Example 6: Use DSN1LOGP on a single member of a data sharing group. The following example shows how to extract log information that pertains to the table space that is identified by DBID X'112' and OBID X'1D' from a single member of a data sharing group.

```
//STEP1 EXEC PGM=DSN1LOGP
//STEPLIB DD  DSN=PDS containing DSN1LOGP
//SYSPRINT SYSOUT=A
//SYSABEND SYSOUT=A
//M01BSDS DD DSN=DSNDB0G.DB1G.BSDS01,DISP=SHR
//SYSIN DD *
  DATAONLY (YES)
  LRSNSTART (A7951A001AD5) LRSNEND (A7951A003B6A)
  DBID (112) OBID(1D)
/*
```

DSN1LOGP output

One intended use of this utility is to aid in determining and correcting system problems. When diagnosing DB2, you might need to refer to licensed documentation to interpret output from this utility. For more information about diagnosing problems, see *DB2 Diagnosis Guide and Reference*.

Reviewing DSN1LOGP output

With the SUMMARY option, you can produce a summary report, a detail report, or both. You can also use the CHECK(DATA) option to produce a summary and detail report of page regression errors.

Figure 138 on page 761 shows a sample of the summary report. Figure 139 on page 762 shows a sample of the detail report. Figure 140 on page 764 shows a sample of data propagation information from a summary report. A description of the output precedes each sample.

Description of the summary report

The summary report in Figure 138 on page 761 contains a summary of completed events, consisting of an entry for each completed unit of work. Each entry shows, among other information, the start time, user, and all page sets that were modified.

The summary report is divided into two distinct sections:

- The first section is headed by the following message:
  `DSN1150I SUMMARY OF COMPLETED EVENTS`
- The second section is headed by the following message:
  `DSN1157I RESTART SUMMARY`

The first section lists all completed units of recovery (URs) and checkpoints within the range of the log that is scanned. Events are listed chronologically, with URs listed according to when they were completed and checkpoints listed according to when the end of the checkpoint was processed. The page sets that are changed by each completed UR are listed. If a log record that is associated with a UR is unavailable, the attribute INFO=PARTIAL is displayed for the UR. Otherwise, the UR is marked INFO=COMPLETE. A log record that is associated with a UR is unavailable if the range of the scanned log is not large enough to contain all records for a given UR.

The DISP attribute can be one of the following values: COMMITTED, ABORTED, INFLIGHT, IN-COMMIT, IN-ABORT, POSTPONED ABORT, or INDOUBT. The DISP attributes COMMITTED and ABORTED are used in the first section; the remaining attributes are used in the second section.

The list in the second section shows the work that is required of DB2 at restart as it is recorded in the log that you specified. If the log is available, the checkpoint that is to be used is identified, as is each outstanding UR, together with the page sets it changed. Each page set with pending writes is also identified, as is the earliest log record that is required to complete those writes. If a log record that is associated with a UR is unavailable, the attribute INFO=PARTIAL is displayed, and the identification of modified page sets is incomplete for that UR.

```
DSN1212I DSN1LGRD FIRST LOG LRSN ENCOUNTERED AA526968220D

================================================================================
DSN1150I SUMMARY OF COMPLETED EVENTS

DSN1151I DSN1LPRT  MEMBER=V81B  UR CONNID=V81B   CORRID=021.OPNLGR00  AUTHID=SYSOPR  PLAN=SYSTEM
   START DATE=94.347 TIME=11:15:22  DISP=COMMITTED   INFO=COMPLETE
   STARTRBA=00000000E570  ENDRBA=00000000EB64  STARTLRSN=AA52696B1269  ENDLRSN=AA526999D14D  NID=*
   LUWID=USIBMSY.SYEC1B.AA52696825CE.0001  COORDINATOR=*
   PARTICIPANTS=*
   DATA MODIFIED:
        DATABASE=0001=DSNDB01   PAGE SET=00CF=SYSLGRNX
        DATABASE=0001=DSNDB01   PAGE SET=0087=DSNLLX01
        DATABASE=0001=DSNDB01   PAGE SET=0086=DSNLLX02

DSN1151I DSN1LPRT  MEMBER=V81B  UR CONNID=V81B   CORRID=021.OPNLGR00  AUTHID=SYSOPR  PLAN=SYSTEM
   START DATE=94.347 TIME=11:16:14  DISP=COMMITTED   INFO=COMPLETE
   STARTRBA=00000000ECFC  ENDRBA=00000000F20A  STARTLRSN=AA52699C97A9  ENDLRSN=AA52699CADC5  NID=*
   LUWID=USIBMSY.SYEC1B.AA52699C9508.0001  COORDINATOR=*
   PARTICIPANTS=*
   DATA MODIFIED:
        DATABASE=0001=DSNDB01   PAGE SET=00CF=SYSLGRNX
        DATABASE=0001=DSNDB01   PAGE SET=0087=DSNLLX01
        DATABASE=0001=DSNDB01   PAGE SET=0086=DSNLLX02
....

DSN1213I DSN1LGRD LAST LOG LRSN ENCOUNTERED AA527C9B8392

DSN1214I NUMBER OF LOG RECORDS READ 0000000000004991

================================================================================
DSN1157I RESTART SUMMARY

DSN1153I DSN1LSIT CHECKPOINT  MEMBER=V81B
   STARTRBA=000000068CD3 ENDRBA=00000006CAED STARTLRSN=AA527AA809DF ENDLRSN=AA527AA829F4
   DATE=94.347  TIME=12:32:29

DSN1162I DSN1LPRT  MEMBER=V81C  UR CONNID=BATCH CORRID=S5529927 AUTHID=ADMF001 PLAN=PLNFW543
   START DATE=94.347 TIME=12:41:04  DISP=INFLIGHT    INFO=COMPLETE
   STARTRBA=000000016000  STARTLRSN=AA527C9278DF  NID=*
   LUWID=USIBMSY.SYEC1C.AA527C22E283.0001  COORDINATOR=*
   PARTICIPANTS=*
   DATA MODIFIED:
        DATABASE=0113=DBFW5401  PAGE SET=0002=TPFW5401
        DATABASE=0113=DBFW5401  PAGE SET=0005=IPFW5401

DSN1162I DSN1LPRT MEMBER=V81A   UR CONNID=BATCH CORRID=S5529925 AUTHID=ADMF001 PLAN=PLNFW541
   START DATE=94.347 TIME=12:41:04  DISP=INFLIGHT   INFO=COMPLETE
   STARTRBA=000001F9A3C1  STARTLRSN=AA527C92E419  NID=*
   LUWID=USIBMSY.SYEC1DB2.AA527C1D674B.0001  COORDINATOR=*
   PARTICIPANTS=*
   DATA MODIFIED:
        DATABASE=0113=DBFW5401  PAGE SET=0002=TPFW5401
 ...
 DSN1160I DATABASE WRITES PENDING:
          DATABASE=0001=DSNDB01    PAGE SET=0046=DSNLUX02   START=000000068CD3
          DATABASE=0001=DSNDB01    PAGE SET=0044=DSNLUX01   START=000000068CD3
 ...
          DATABASE=0006=DSNDB06    PAGE SET=0076=DSNUCX01   START=000000068CD3
          DATABASE=0006=DSNDB06    PAGE SET=0072=DSNUCH01   START=000000068CD3
 ...
```

Figure 138. Sample DSN1LOGP summary report

Description of the detail report

The detail report in Figure 139 includes the following records:

- Redo and undo log records
- System events log records, including begin and end checkpoint records, begin current status rebuild records, and begin forward and backward recovery records
- Page set control log records, including open and close page set log records, open and close data set log records, set write, reset write, and page set write log records
- UR control log records for the complete or incomplete unit of recovery

You can reduce the volume of the detail log records by specifying one or more of the optional keywords that are listed under "Syntax and options of the DSN1LOGP control statement" on page 748.

```
DSN1212I DSN1LGRD FIRST LOG  RBA ENCOUNTERED 00000335916E

0000033591D4  MEMBER(M01     )  LRSN(AB62536BE583)  DBID(0006)  OBID(00B2)
     TYPE(PAGE SET CONTROL)  SUBTYPE(PAGE SET STATUS RECORD)
     *LRH* 00660066 00020009 0E800000 00000000 00000335 916E0126 00000335 916EAB62
           536BE583 0001
     0000  000600B2 C4E2D5C4 C2F0F640 C4E2D5E3 D5E7F0F1 00010000 92018000 00000334
     0020  EC3AAB62 5260AB0B 00000000 00000000 00000000 00000000 00000000 00000000

0000000109E2  MEMBER(M02     )  LRSN(AB6253746CE3)  DBID(0113)  OBID(0008)
     TYPE(PAGE SET CONTROL)  SUBTYPE(PAGE SET OPEN)
     *LRH* 00A0006E 00020001 0E800000 00000000 00000000 00000126 00000000 0000AB62
           53746CE3 0002
     0000  01130008 6C010100 00000005 0040C4C2 C6E6F0F0 F1F1C9C3 C6E6F0F0 F0F10001
     0020  00060000 10009201 00130020 00000000 00000000 00000000 00000000 00000000
     0040  00000000 00000010 00010000 00000000 00000000 00000000 00000000 00000000
     0060  00AB624B 192CEEAB 624B4783 F8000000 0000C4E2 D5C3F4F1 F040

000000010A82  MEMBER(M02     )  URID(000000010A82)  LRSN(AB6253747801)
     TYPE(UR CONTROL)  SUBTYPE(BEGIN UR)
     *LRH* 009000A0 00200001 03800000 00010A82 00000000 00000126 00000000 0000AB62
           53747801 0002
     0000  00010000 0000D000 00000000 00000700 0000D4F0 F0F0F1F0 F2F54040 4040D7C6
     0020  E5E3F0F0 F340AB62 537477FC B803C4E2 D5E3C5D7 F340C2C1 E3C3C840 4040C2C1
     0040  E3C3C840 40400000 00000000 0000001A 0001E4E2 C9C2D4E2 E840E2E8 C5C3F1C4
     0060  4040AB62 5362554A 0001

000000010B12  MEMBER(M02     )  URID(000000010A82)  LRSN(AB6253747807)
     TYPE( UNDO )  SUBTYPE(SAVEPOINT)
     *LRH* 002F0090 22000014 0E800000 00010A82 00000001 0A820126 00000001 0A82AB62
           53747807 0002
     0000  00E7D9E4 C9000000 02

000000010B42 MEMBER(M02    )   URID(000000010A82)   LRSN(AB625374780E)  DBID(0113)  OBID(0008)
             PAGE(00000003)
     TYPE( UNDO  REDO )  SUBTYPE(TYPE 2 INDEX UPDATE)  CLR(NO)  PROCNAME(DSNKDLE )
     *LRH* 0053002F 06000019 0E800000 00010A82 00000001 0B120126 00000001 0B12AB62
           5374780E 0002
     *LG** 84011300 08000003 63000000 00000000 0000
     0000  001B3000 00B40001 00000201 000A0000 02C5C5F0 F6C1C1D4 F3F1C1
```

Figure 139. Sample DSN1LOGP detail report (Part 1 of 2)

```
000000010B94  MEMBER(M02     )  URID(000000010A82)  LRSN(AB6253747CEF)  DBID(0113)  OBID(0008)
              PAGE(00000003)
     TYPE( UNDO  REDO )  SUBTYPE(TYPE 2 INDEX UPDATE)  CLR(NO)  PROCNAME(DSNKINSL)
     *LRH* 00530053 06000019 0E800000 00010A82 00000001 0B420126 00000001 0B42AB62
           53747CEF 0002
     *LG** 04011300 08000003 64000000 00000000 0000
     0000  001B1000 00B30001 00000201 000A2000 00C5C5F0 F6C1C1D7 D7D3F4
......
00000001138E  MEMBER(M02     )  URID(0000000110A0)  LRSN(AB62537B4931)
     TYPE(UR CONTROL)  SUBTYPE(BEGIN COMMIT1)
     *LRH* 005C0053 00200002 03800000 000110A0 00000001 133B0126 00000001 133BAB62
           537B4931 0002
     0000  00020000 00004000 00000000 00000700 0000F0F2 F14BD6D7 D5D3C7D9 F0F04040
     0020  40404040 40400000 00000000 00000000 00000000 0000

0000000113EA  MEMBER(M02     )  URID(0000000110A0)  LRSN(AB62537B4940)
     TYPE(UR CONTROL)  SUBTYPE(PHASE 1 TO 2)
     *LRH* 0034005C 0020000C 03800000 000110A0 00000001 138E0126 00000001 138EAB62
           537B4940 0002
     0000  00020000 00004000 00000000 0000

0000033685DE  MEMBER(M01     )  LRSN(AB6254D9A231)  DBID(0001)  OBID(001F)
     TYPE( CHECKPOINT)  SUBTYPE(DBE TABLE WITH EXCEPTION DATA)
     *LRH* 0061003E 2100001D 0E800000 00000000 00000336 85A00126 00000336 85A0AB62
           54D9A231 0001
     0000  00000000 C4E2D5C4 C2F0F140 C4C2C4F0 F1404040 0001001F 00000000 00000000
     0020  00000000 00000000 00000000 00000000 00000000 00000000 000000

00000336863F  MEMBER(M01     )  LRSN(AB6254D9A237)  DBID(0001)  OBID(001F)
     TYPE( CHECKPOINT)  SUBTYPE(DBE TABLE WITH PIECE DATA)
     *LRH* 01F60061 2100001E 0E800000 00000000 00000336 85DE0126 00000336 85DEAB62
           54D9A237 0001
     0000  00000100 1FC4E2D5 C4C2F0F1 40C4C2C4 F0F14040 40000000 0020FFFF FFFFFFFF
     0020  00000000 00000000 0000006C 00000090 FFFFFFFF 00000000 00000000 00FFFFFF
     0040  FF000000 00000000 0000FFFF FFFF0000 00000000 000000FF FFFFFF00 00000000
     0060  00000000 FFFFFFFF 00000000 00000000 00FFFFFF FF000000 00000000 0000FFFF
     0080  FFFF0000 00000000 000000FF FFFFFF00 00000000 00000000 FFFFFFFF 00000000
     00A0  00000000 00FFFFFF FF000000 00000000 0000FFFF FFFF0000 00000000 000000FF
     00C0  FFFFFF00 00000000 00000000 FFFFFFFF 00000000 00000000 00FFFFFF FF000000
     00E0  00000000 0000FFFF FFFF0000 00000000 000000FF FFFFFF00 00000000 00000000
     0100  FFFFFFFF 00000000 00000000 00FFFFFF FF000000 00000000 0000FFFF FFFF0000
     0120  00000000 000000FF FFFFFF00 00000000 00000000 FFFFFFFF 00000000 00000000
     0140  00FFFFFF FF000000 00000000 0000FFFF FFFF0000 00000000 000000FF FFFFFF00
     0160  00000000 00000000 FFFFFFFF 00000000 00000000 00FFFFFF FF000000 00000000
     0180  0000FFFF FFFF0000 00000000 000000FF FFFFFF00 00000000 00000000 FFFFFFFF
     01A0  00000000 00000000 00FFFFFF FF000000 00000000 0000FFFF FFFF0000 00000000
     01C0  000000FF FFFFFF00 00000000 00000000
 ...
000000057EA2  MEMBER(M02     )  LRSN(AB62564FF606)  DBID(0113)  OBID(000A)
     TYPE(PAGE SET CONTROL)  SUBTYPE(PAGE SET WRITE)
     *LRH* 009C002A 00020007 0E800000 00000000 00000005 7E780126 00000005 7E78AB62
           564FF606 0002
     0000  0000F5D7 C3D60113 000AC4C2 C6E6F0F0 F1F1C9E4 C6E6F0F0 F0F20000 00000000
     0020  00000000 0004D98E 00000004 D98E0000 00000000 11040000 00000000 03000003
     0040  AB62553F 98780000 00000000 04000004 AB62553F 930C0000 00000000 05000005
     0060  AB62553F 95C30000 00000000 06000006 AB62553F 9855

000000057F3E  MEMBER(M02     )  LRSN(AB62564FFFBA)  DBID(0113)  OBID(000A)
     TYPE(PAGE SET CONTROL)  SUBTYPE(PAGE SET CLOSE)
     *LRH* 002A009C 00020003 0E800000 00000000 00000005 7EA20126 00000005 7EA2AB62
           564FFFBA 0002
     0000  0113000A


DSN1213I DSN1LGRD LAST LOG  RBA ENCOUNTERED 00000337A000


DSN1214I NUMBER OF LOG RECORDS READ 0000000000004661
```

Figure 139. Sample DSN1LOGP detail report (Part 2 of 2)

Description of data propagation information in the summary report

The sample output in Figure 140 shows information from the DSN1LOGP summary report about log records of changes to DB2 tables that were defined with DATA CAPTURE CHANGES.

The fields show the following information:

- START RBA and END RBA show the first and last RBAs that are captured for the unit of recovery that was not retrieved. The range that the start and end RBA encompass can include one or all of the SQL statements within the scope of the unit of recovery.
- TABLE LIST OVERFLOW indicates whether more than 10 distinct data capture table IDs were updated by this unit of recovery. This example indicates that no overflow occurred.
- LR WRITTEN shows the number of written log records that represented changes to tables that were defined for data capture and were available to the DB2CDCEX routine. Recursive SQL changes from DB2CDCEX and changes from other attachments that are not associated with DB2CDCEX are not included. If you receive a value of 2147483647, an overflow occurred and the count is not valid.
- LR RETRIEVED is the number of captured RBAs that were retrieved by DB2CDCEX. If you receive a value of 2147483647, an overflow occurred and the count is not valid.
- LR NOT RETRIEVED is the difference between the number of written log records (LR WRITTEN) and the number of retrieved log records (LR RETRIEVED). The following example output shows that four log records were written, and none were retrieved.

```
DATA PROPAGATION INFORMATION:
     START RBA=000004A107F4       END RBA=000004A10A5C           TABLE LIST OVERFLOW=NO
     LR WRITTEN=0000000000000004  LR RETRIEVED=0000000000000000  LR NOT RETRIEVED=0000000000000004
     DATABASE=0112=DBCS1701       PAGESET=0002=TSCS1701          TABLE OBID=0005
```

Figure 140. Sample data propagation information from the summary report

Description of the report on page regression errors

DSN1LOGP reports page regression errors when you specify the CHECK(DATA) option. The value of the SUMMARY option determines whether the utility creates a detail report, a summary report, or both.

A detail report contains the following information for each page regression error:

- DBID
- OBID
- Page number
- Current LRSN or RBA
- Member name
- Previous level
- Previous update
- Date
- Time

A summary report contains the total number of page regressions that the utility found as well as the following information for each table space in which it found page regression errors:

- Database name
- Table space name
- DBID
- OBID

If no page regression errors are found, DSN1LOGP outputs a single message that no page regression errors were found.

The sample output in Figure 141 shows detail and summary reports when page regression errors are found.

```
DSN1212I  DSN1LGRD FIRST LOG RBA ENCOUNTERED 5182C4758000

DSN1212I  DSN1LGRD FIRST LOG LRSN ENCOUNTERED B7A829006D13

DSN1191I:
-------------------------------------------------------------------------------
                DETAIL REPORT OF PAGE REGRESSION ERRORS
-------------------------------------------------------------------------------
DBID OBID   PAGE#    CURRENT    MEMBER  PREV-LEVEL   PREV-UPDATE   DATE   TIME
---- ---  -------- ------------ ------- -----------  ------------ ------ --------
0001 OOCF OOOO132F B7A83F071892 0002    84A83BBEE81F B7A83C6042DF 02.140 15:29:20
0001 OOCF 000086C2 B7A84BD4C3E5 0003    04A83BC42C58 B7A83C61D53E 02.140 18:01:13
0006 0009 00009DBF B7A8502A39F4 0002    04A83BC593B6 B7A83C669743 02.140 18:20:37

DSN1213I DSN1LGRD LAST LOG RBA ENCOUNTERED 51830AC57F47


DSN1213I DSN1LGRD LAST LOG LRSN ENCOUNTERED B7A8568367E6


DSN12141 NUMBER OF LOG RECORDS READ 0000000007816406
DSN1194I:
-------------------------------------------------------------------------------
                SUMMARY REPORT OF PAGE REGRESSION ERRORS
-------------------------------------------------------------------------------
                DATABASE SPACENAM DBID OBID #PG REGRESSIONS
                -------- -------- ---- ---- ---------------
                DSNDB01  SYSLGRNX 0001 00CF 00000002
                DSNDB06  SYSDBASE 0006 0009 00000001
-------------------------------------------------------------------------------
DSN1197I TOTAL PAGES CHECKED FOR REGRESSION = 00312927
:
:
```

Figure 141. Sample DSN1LOGP detail and summary reports for page regression errors.

Interpreting error codes

When an error occurs, DSN1LOGP formats a reason code from the DB2 stand-alone log service in the SYSPRINT output. For information about the stand-alone log service and the reason codes that it issues, see Appendix C (Volume 2) of *DB2 Administration Guide*.

DSN1LOGP can abnormally terminate with a user abend code of X'099'. DSN1LOGP finds the corresponding abend reason code in register 15 (at the time of error).

Chapter 42. DSN1PRNT

With the DSN1PRNT stand-alone utility, you can print:
- DB2 VSAM data sets that contain table spaces or index spaces (including dictionary pages for compressed data)
- Image copy data sets
- Sequential data sets that contain DB2 table spaces or index spaces

Note: A DB2 VSAM data set is a single piece of a nonpartitioned table space or index, or a single partition of a partitioned table space or index. The input must be a single z/OS sequential or VSAM data set. Concatenation of input data sets is not supported.

Using DSN1PRNT, you can print hexadecimal dumps of DB2 data sets and databases. If you specify the FORMAT option, DSN1PRNT formats the data and indexes for any page that does not contain an error that would prevent formatting. If DSN1PRNT detects such an error, it prints an error message just before the page and dumps the page without formatting. Formatting resumes with the next page.

Compressed records are printed in compressed format.

DSN1PRNT is especially useful when you want to identify the contents of a table space or index. You can run DSN1PRNT on image copy data sets and on table spaces and indexes. DSN1PRNT accepts an index image copy as input when you specify the FULLCOPY option.

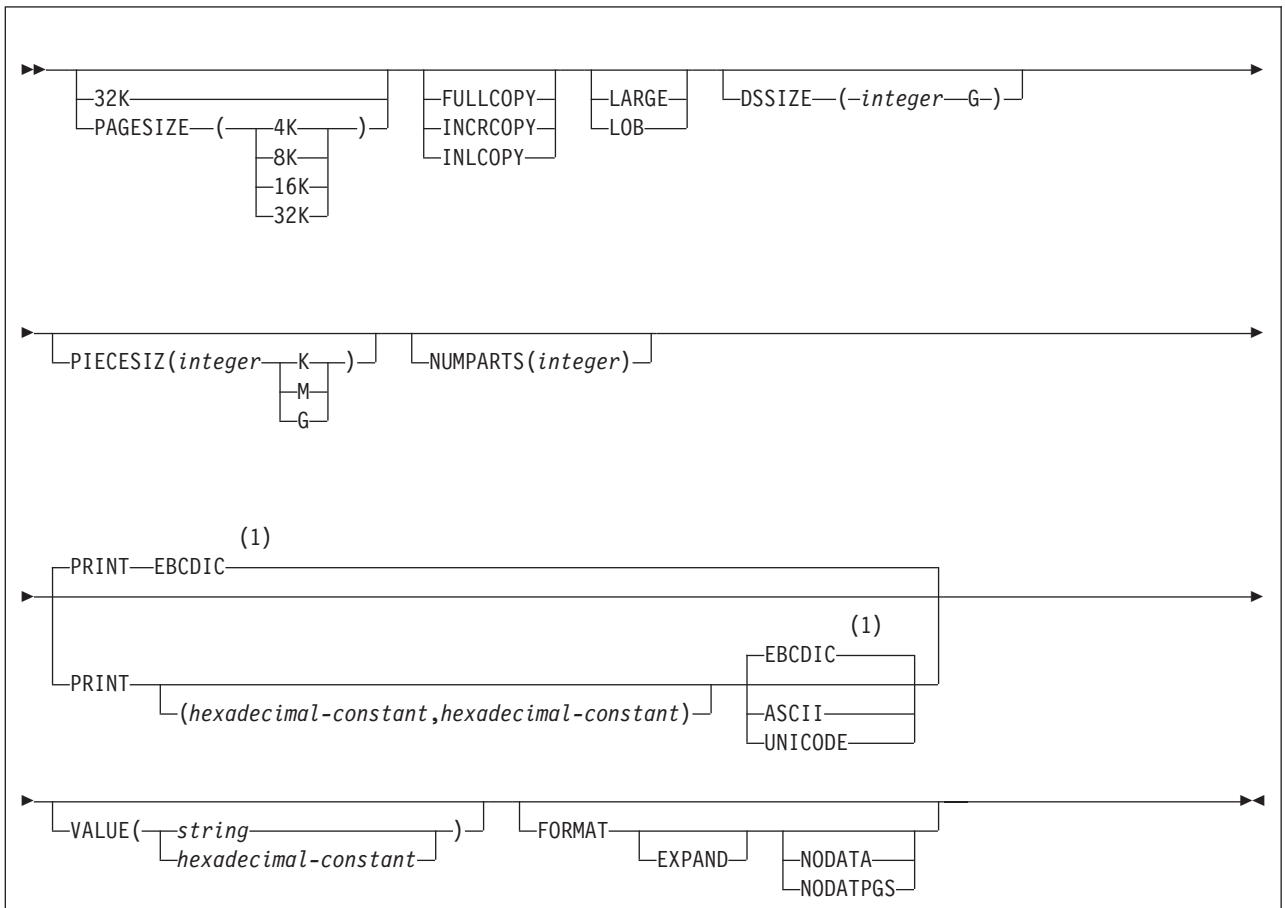
You cannot run DSN1PRNT on concurrent copies.

DSN1PRNT is compatible with LOB table spaces, when you specify the LOB keyword and omit the INLCOPY keyword.

The following topics provide additional information:
- "Syntax and options of the DSN1PRNT control statement" on page 768
- "Before running DSN1PRNT" on page 773
- "Sample DSN1PRNT control statements" on page 775
- "DSN1PRNT output" on page 776

Syntax and options of the DSN1PRNT control statement

DSN1PRNT syntax diagram



Notes:

1 EBCDIC is not necessarily the default if the first page of the input data set is a header page. If the first page is a header page, DSN1PRNT uses the format information in the header page as the default format.

Option descriptions

To run DSN1PRNT, specify one or more of the following parameters on the EXEC statement.

Important: If you specify more than one parameter:
- Separate them by commas (no blanks).
- Specify them in any order.

32K
Specifies that the SYSUT1 data set has a 32-KB page size. If you specify this option and the SYSUT1 data set does not have a 32-KB page size, DSN1COPY might produce unpredictable results.

The recommended option for performance is **PAGESIZE(32K)**.

PAGESIZE Specifies the page size of the input data set that is defined by SYSUT1. Available page size values are 4K, 8K, 16K, or 32K. If you specify an incorrect page size, DSN1PRNT might produce unpredictable results.

If you do not specify the page size, DSN1PRNT tries to determine the page size from the input data set if the first page of the input data set is a header page. DB2 issues an error message if DSN1PRNT cannot determine the input page size. This might happen if the header page is not in the input data set, or if the page size field in the header page contains an invalid page size.

FULLCOPY Specifies that a DB2 full image copy (not a DFSMSdss concurrent copy) of your data is to be used as input. If this data is partitioned, you also need to specify the NUMPARTS parameter to identify the number and length of the partitions. If you specify FULLCOPY without including a NUMPARTS specification, DSN1PRNT assumes that the input file is not partitioned.

The FULLCOPY parameter must be specified when you use an image copy as input to DSN1PRNT. Omitting the parameter can cause error messages or unpredictable results.

INCRCOPY Specifies that an incremental image copy of the data is to be used as input. If the data is partitioned, also specify NUMPARTS to identify the number and length of the partitions. If you specify INCRCOPY without NUMPARTS, DSN1PRNT assumes that the input file is not partitioned.

The INCRCOPY parameter must be specified when you use an incremental image copy as input to DSN1PRNT. Omitting the parameter can cause error messages or unpredictable results.

INLCOPY Specifies that the input data is to be an inline copy data set.

When you use DSN1PRNT to print a page or a page range from an inline copy that is produced by LOAD or REORG, DSN1PRNT prints all instances of the pages. The last instance of the printed page or pages is the last one that is created by the utility.

LARGE Specifies that the input data set is a table space that was defined with the LARGE option, or an index on such a table space. If you specify LARGE, DB2 assumes that the data set has a 4-GB boundary. The recommended method of specifying a table space that was defined with the LARGE option is **DSSIZE(4G)**.

If you omit the LARGE or DSSIZE(4G) option when it is needed, or if you specify LARGE for a table space that was not defined with the LARGE option, the results from DSN1PRNT are unpredictable.

If you specify LARGE, you cannot specify LOB or DSSIZE.

LOB Specifies that the SYSUT1 data set is a LOB table space. You cannot specify the INLCOPY option with the LOB parameter.

DB2 attempts to determine if the input data set is a LOB data set. If you specify the LOB option but the data set is not a LOB data set, or if you omit the LOB option but the data set is a LOB data set, DB2 issues an error message and DSN1PRNT terminates.

If you specify LOB, you cannot specify LARGE.

DSSIZE(*integer* **G)**
Specifies the data set size, in gigabytes, for the input data set. If you omit the DSSIZE keyword or the LARGE keyword, DSN1PRNT assumes the appropriate default input data set size that is listed in Table 141.

Table 141. Default input data set sizes

Object	Default input data set size (in GB)
Non-LOB linear table space or index	2
LOB	4
Partitioned table space or index with NUMPARTS = 1-16	4
Partitioned table space or index with NUMPARTS = 17-32	2
Partitioned table space or index with NUMPARTS = 33-64	1
Partitioned table space or index with NUMPARTS >64	4

integer must match the DSSIZE value that was specified when the table space was defined.

If you omit DSSIZE and the data set is not one of the default sizes, the results from DSN1PRNT are unpredictable.

If you specify DSSIZE, you cannot specify LARGE.

PIECESIZ(*integer***)**
Specifies the maximum piece size (data set size) for nonpartitioned indexes. The value that you specify must match the value that is specified when the secondary index was created or altered.

The defaults for PIECESIZ are 2G (2 GB) for indexes that are backed by non-large table spaces and 4G (4 GB) for indexes that are backed by table spaces that were defined with the LARGE option. This option is required if a print range is specified and the piece size is not one of the default values. If PIECESIZ is omitted and the index is backed by a table space that was defined with the LARGE option, the LARGE keyword is required for DSN1PRNT.

The subsequent keyword K, M, or G, indicates the units of the value that is specified in *integer*.

K Indicates that the *integer* value is to be multiplied by 1 KB to specify the maximum piece size in bytes. *integer* must be either 256 or 512.

M Indicates that the *integer* value is to be multiplied by 1 MB to specify the maximum piece size in bytes. *integer* must be a power of 2, between 1 and 512.

G Indicates that the *integer* value is to be multiplied by 1 GB to specify the maximum piece size in bytes. *integer* must be 1, 2, or 4.

Valid values for piece size are:
- 1 MB or 1 GB
- 2 MB or 2 GB

- 4 MB or 4 GB
- 8 MB
- 16 MB
- 32 MB
- 64 MB
- 128 MB
- 256 KB or 256 MB
- 512 KB or 512 MB

NUMPARTS(*integer***)**

Specifies the number of partitions that are associated with the input data set. NUMPARTS is required if the input data set is partitioned. When you use DSN1PRNT to copy a data-partitioned secondary index, specify the number of partitions in the index.

Valid specifications range from 1 to 4096. DSN1PRNT uses this value to help locate the first page in a range that is to be printed. If you omit NUMPARTS or specify it as 0, DSN1PRNT assumes that your input file is not partitioned. If you specify a number greater than 64, DSN1PRNT assumes that the data set is for a partitioned table space that was defined with the LARGE option, even if the LARGE keyword is not specified for DSN1PRNT.

DSN1PRNT cannot always validate the NUMPARTS parameter. If you specify it incorrectly, DSN1PRNT might print the wrong data sets or return an error message that indicates that an unexpected page number was encountered.

PRINT(*hexadecimal-constant,hexadecimal-constant***)**

Causes the SYSUT1 data set to be printed in hexadecimal format on the SYSPRINT data set. This option is the default for DSN1PRNT.

You can specify the PRINT parameter with or without page range specifications. If you do not specify a range, all pages of the SYSUT1 are printed. If you want to limit the range of pages that are printed, you can do so by indicating the beginning and ending page numbers with the PRINT parameter or, if you want to print a single page, by indicating only the beginning page. In either case, your range specifications must be from one to eight hexadecimal characters in length.

The following example shows how to code the PRINT parameter if you want to begin printing at page X'2F0' and to stop at page X'35C':

```
PRINT(2F0,35C)
```

Note that the actual size of a 4-GB DB2 data set that is full is `4G - 256 x 4KB`. This size also applies to data sets that are created with a DFSMS data class that has extended addressability. When calculating the print range of pages in a non-first data set of a multiple data set linear table space or index with 4G DSSIZE or PIECESIZ, use the actual data set size.

The relationship between the page size and the number of pages in a 4-GB data set is shown in Table 142 on page 772.

Table 142. Relationship between page size and the number of pages in a 4-GB data set

Page size	Number of pages
4 KB	X'FFF00'
8 KB	X'7FF80'
16 KB	X'3FFC0'
32 KB	X'1FFE0'

For example, if PAGESIZE is 4 KB, the page number of the first page of the third data set is 2*FFF00 = 1FFE00.

To print only the header page for a nonpartitioned table space, specify PRINT(0). For guidance on specifying page numbers for partitioned table spaces, see "Using VERIFY with REPLACE and DELETE operations" on page 517 and "Data organization" in *DB2 Diagnosis Guide and Reference*.

You can indicate the format of the row data in the PRINT output by specifying EBCDIC, ASCII, or UNICODE. The part of the output that is affected by these options is in bold in Figure 142.

```
RECORD:  XOFFSET='0014'X  PGSFLAGS='00'X  PGSLTH=65  PGSLTH='0041'X  PGSOBD='0003'X  PGSBID='01'X
C5C5F0F6 C1404040 40404040 F1F34040 40C1E2D6 F1F3F5E7 40404040 40404040   EE06A       13   AS0135X
C1C6F3F1 C587C6F0 01800000 14199002 01174522 00000080 000000              AF31E.F0...................

RECORD:  XOFFSET='0055'X  PGSFLAGS='00'X  PGSLTH=65  PGSLTH='0041'X  PGSOBD='0003'X  PGSBID='02'X
C5C5F0F6 C1404040 40404040 F1F34040 40C1E2D6 F1F3F5E7 40404040 40404040   EE06A       13   AS0135X
C1C6F5F2 D487C5F0 09800000 78199002 01174522 00000080 000000              AF52M.E0...................
```

Figure 142. The part of the DSN1PRNT FORMAT output that is affected by the EBCDIC, ASCII, and UNICODE options

EBCDIC
Indicates that the row data in the PRINT output is to be displayed in EBCDIC. The **default** is **EBCDIC** if the first page of the input data set is not a header page.

If the first page is a header page, DSN1PRNT uses the format information in the header page as the default format. However, if you specify EBCDIC, ASCII, or UNICODE, that format overrides the format information in the header page. The unformatted header page dump is always displayed in EBCDIC, because most of the fields are in EBCDIC.

ASCII
Indicates that the row data in the PRINT output is to be displayed in ASCII. Specify ASCII when printing table spaces that contain ASCII data.

UNICODE
Indicates that the row data in the PRINT output is to be displayed in Unicode. Specify UNICODE when printing table spaces that contain Unicode data.

VALUE Causes each page of the input data set SYSUT1 to be scanned for the character string that you specify in parentheses following the VALUE parameter. Each page that contains that character string is then printed in SYSPRINT. You can specify the VALUE parameter in conjunction with any of the other DSN1PRNT parameters.

(*string***)**
Can consist of from 1 to 20 alphanumeric EBCDIC characters. For non-EBCDIC characters, use hexadecimal characters.

(*hexadecimal-constant***)**
Consists of from 2 to 40 hexadecimal characters. You must specify two apostrophe characters before and after the hexadecimal character string.

If, for example, you want to search your input file for the string '12345', your JCL should look like the following JCL:

```
//STEP1 EXEC PGM=DSN1PRNT,PARM='VALUE(12345)'
```

Alternatively, you might want to search for the equivalent hexadecimal character string, in which case your JCL should look like the following JCL:

```
//STEP1 EXEC PGM=DSN1PRNT,PARM='VALUE(''F1F2F3F4F5'')'
```

FORMAT Causes the printed output to be formatted. Page control fields are identified, and individual records are printed. Empty fields are not displayed.

EXPAND
Specifies that the data is compressed and causes DSN1PRNT to expand it before formatting. The FORMAT EXPAND function is not supported for incremental and inline copies.

NODATA
Suppresses printing of table row data. The row headers are formatted and printed. This keyword is ignored for indexes. Specify NODATA to reduce the volume of the output when the contents of the rows are not important.

NODATPGS
Suppresses all data pages of a table space. This keyword is ignored for indexes. Specify NODATPGS to format and print only non-data pages to reduce the volume of the output when only certain page types are of interest (for example, LOB space map pages). Alternatively, you can specify NODHDR.

DSN1PRNT cannot format a leaf or nonleaf page for an index page set that contains keys with altered columns. When it encounters this situation, DSN1PRNT generates the following message:

```
*KEY WITH ALTERED COLUMN HAS BEEN DETECTED-UNABLE TO FORMAT PAGE*
```

DSN1PRNT generates unformatted output for the page.

FORMAT might generate unformatted output for certain system pages.

Before running DSN1PRNT

This section contains information that you need to know before you run DSN1PRNT.

Environment

Run DSN1PRNT as a z/OS job.

You can run DSN1PRNT even when the DB2 subsystem is not operational. If you choose to use DSN1PRNT when the DB2 subsystem is operational, ensure that the DB2 data sets that are to be printed are not currently allocated to DB2.

To make sure that a data set is not currently allocated to DB2, issue the DB2 STOP DATABASE command, specifying the table spaces and indexes that you want to print.

Authorization required

No special authorization is required. However, if any of the data sets is RACF protected, the authorization ID of the job must have RACF authority.

Control statement

Create the utility control statement for the DSN1PRNT job. See "Syntax and options of the DSN1PRNT control statement" on page 768 for DSN1PRNT syntax and option descriptions.

Required data sets: DSN1PRNT uses the following DD statements:

SYSPRINT Defines the data set that contains output messages from DSN1PRNT and all hexadecimal dump output.

SYSUT1 Defines the input data set. That data set can be a sequential data set or a VSAM data set.

Disposition for this data set must be specified as OLD (DISP=OLD) to ensure that it is not in use by DB2. Specify the disposition for this data set as SHR (DISP=SHR) only in circumstances where the DB2 STOP DATABASE command does not work.

The requested operation takes place only for the specified data set. If the input data set belongs to a linear table space or index space that is larger than 2 GB, or if it is a partitioned table space or index space, you must ensure the correct data set is specified. For example, to print a page range in the second partition of a four-partition table space, specify NUMPARTS(4) and the data set name of the data set in the group of VSAM data sets comprising the table space. The following code shows the data set name:

```
DSN=...A002
```

If you run the online REORG utility with the FASTSWITCH option, verify the data set name before running the DSN1PRNT utility. The fifth-level qualifier in the data set name alternates between I0001 and J0001 when using FASTSWITCH. Specify the correct fifth-level qualifier in the data set name to successfully execute the DSN1PRNT utility. To determine the correct fifth-level qualifier, query the IPREFIX column of SYSIBM.SYSTABLEPART for each data partition or the IPREFIX column of SYSIBM.SYSINDEXPART for each index partition. If the object is not partitioned, use zero as the value for the PARTITION column in your query.

Recommendations

This section contains recommendations for running the DSN1PRNT utility.

Printing with DSN1PRNT instead of DSN1COPY

If you want to print information about a data set, use the DSN1PRNT utility rather than the DSN1COPY utility. DSN1COPY scans the entire SYSUT1 data set, but DSN1PRNT might be able to stop scanning before the end of the data set. Also, the DSN1PRNT utility can write a formatted dump.

Determining page size and DSSIZE

Before using DSN1PRNT, determine the page size and data set size (DSSIZE) for the page set. Use the query in Figure 143 on the DB2 catalog to get the information that you need, in this example for table 'DEPT':

```
SELECT T.CREATOR,T.NAME,S.NAME AS TABLESPACE,S.PARTITIONS,S.PGSIZE,
      CASE S.DSSIZE
      WHEN 0 THEN
                  CASE WHEN S.TYPE = 'O' THEN 4194304
                  ELSE
                       CASE WHEN S.PARTITIONS > 254 THEN
                            CASE WHEN S.PGSIZE = 4  THEN  4194304
                                 WHEN S.PGSIZE = 8  THEN  8388608
                                 WHEN S.PGSIZE = 16 THEN 16777216
                                 WHEN S.PGSIZE = 32 THEN 33554432
                            ELSE NULL
                            END
                            WHEN S.PARTITIONS > 64 THEN 4194304
                            WHEN S.PARTITIONS > 32 THEN 1048576
                            WHEN S.PARTITIONS > 16 THEN 2097152
                            WHEN S.PARTITIONS > 0  THEN 4194304
                       ELSE 2097152
                       END
                  END
      ELSE S.DSSIZE
      END
      AS DSSIZE
      FROM SYSIBM.SYSTABLES T,
           SYSIBM.SYSTABLESPACE S
      WHERE
           T.NAME = 'DEPT' AND
           T.TSNAME = S.NAME;
```

Figure 143. Example SQL query that returns the page size and data set size for the page set.

See “Data sets that REORG INDEX uses” on page 403 for information about determining data set names.

Running DSN1PRNT on encrypted data

DSN1PRNT does not decrypt any encrypted data; the utility displays the data as is.

Sample DSN1PRNT control statements

Example 1: Printing a data set and formatting the output. The following example specifies that the DSN1PRNT utility is to print the data set that is identified by the SYSUT1 DD statement and the output is to be formatted. This data set is to be printed on the data set that is identified by the SYSPRINT DD statement. The fifth-level qualifier in the data set name can be either I0001 or J0001. This example uses I0001.

```
//jobname JOB acct info
//RUNPRNT EXEC PGM=DSN1PRNT,PARM='PRINT,FORMAT'
//STEPLIB DD DSN=prefix.SDSNLOAD
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=DSNCAT.DSNDBC.DSNDB01.SYSUTILX.I0001.A001,DISP=SHR
```

Example 2: Printing a nonpartitioning index with a 64-MB piece size. The following example specifies that DSN1PRNT is to print the first 16 pages of the 61st piece of an nonpartitioned index with a piece size of 64 MB. The pages that are to be printed are identified by the PRINT option. These page values are determined as follows: A data set of size 64 MB contains X'4000' 4-KB pages. Decimal 61 is X'3D'. The page number of the first page of the 61st piece is 4000*(3D-1) = 4000*3C = F0000. To print the last 16 pages of the 61st piece, specify `PARM=(PRINT(F3FF0,F3FFF), ...)`.

The fifth-level qualifier in the data set name can be either I0001 or J0001. This example uses I0001.

```
//PRINT2 EXEC PGM=DSN1PRNT,
//            PARM=(PRINT(F0000,F000F),FORMAT,PIECESIZ(64M))
//SYSUDUMP DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DISP=OLD,DSN=DSNCAT.DSNDBD.MMRDB.NPI1.I0001.A061
```

Example 3: Printing a single page of an image copy. The following example specifies that DSN1PRNT is to print one page of an image copy. The image copy is identified by the SYSUT1 DD statement. The PRINT option specifies that the only page to be printed is X'1'.

```
//STEP2    EXEC PGM=DSN1PRNT,
//         PARM='PRINT(1),FORMAT,INLCOPY'
//STEPLIB  DD DSN=DB2A.SDSNLOAD,DISP=SHR
//SYSUDUMP DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSN=HUHYU205.L1.STEP1.DD2,DISP=SHR
```

Example 4: Printing a partitioned data set. The following example specifies that DSN1PRNT is to print the data set that is identified by the SYSUT1 DD statement. Because this data set is a table space that was defined with the LARGE option, the DSSIZE(4G) option is specified in the parameter list for DSN1PRNT. You could specify the LARGE option in this list instead, but specifying DSSIZE(4G) is recommended. This input table space has 260 partitions, as indicated by the NUMPARTS option.

```
//RUNPRNT1 EXEC PGM=DSN1PRNT,
//    PARM='DSSIZE(4G),PRINT,NUMPARTS(260),FORMAT'
//STEPLIB  DD DSN=DB2A.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSN=DSNCAT.DSNDBC.DBOM0301.TPOM0301.I0001.A259,DISP=SHR
/*
```

DSN1PRNT output

One intended use of this utility is to aid in determining and correcting system problems. When diagnosing DB2, you might need to refer to licensed documentation to interpret output from this utility. For more information about diagnosing problems, see *DB2 Diagnosis Guide and Reference*.

Chapter 43. DSN1SDMP

Under the direction of IBM Software Support, use the IFC selective dump (DSN1SDMP) utility to:

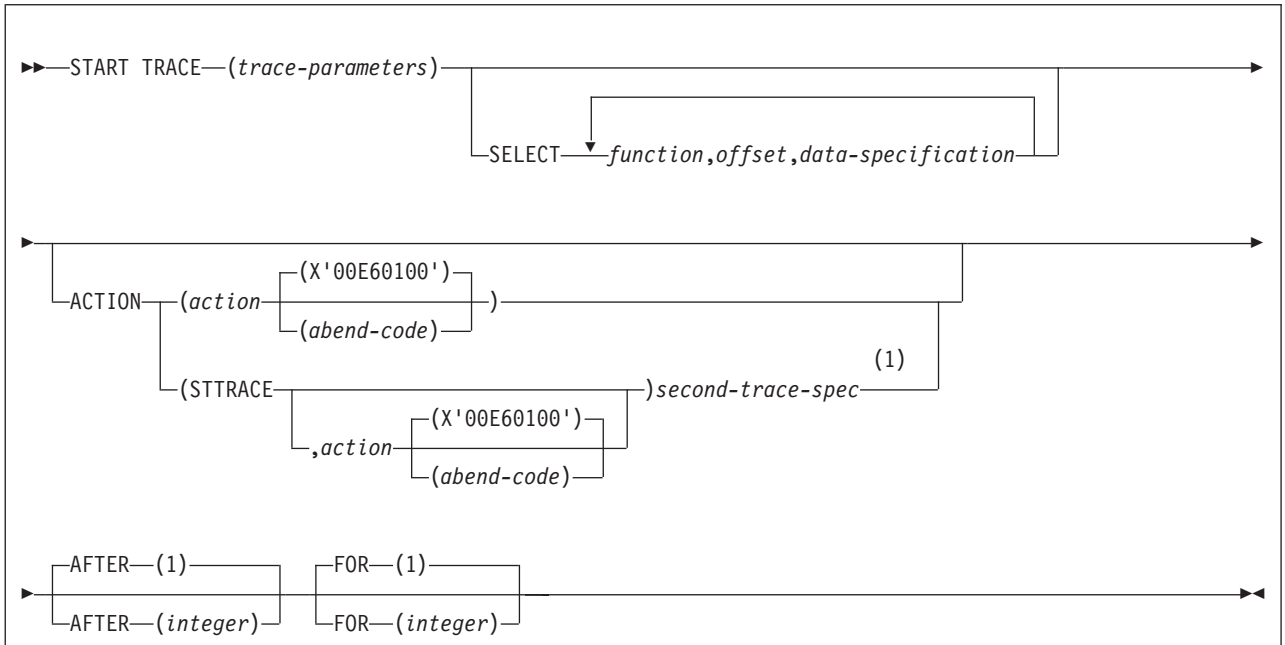
- Force dumps when selected DB2 trace events occur.
- Write DB2 trace records to a user-defined z/OS data set.

For information about the format of trace records, see Appendix D (Volume 2) of *DB2 Administration Guide*.

The following topics provide additional information:

Syntax and options of the DSN1SDMP control statement

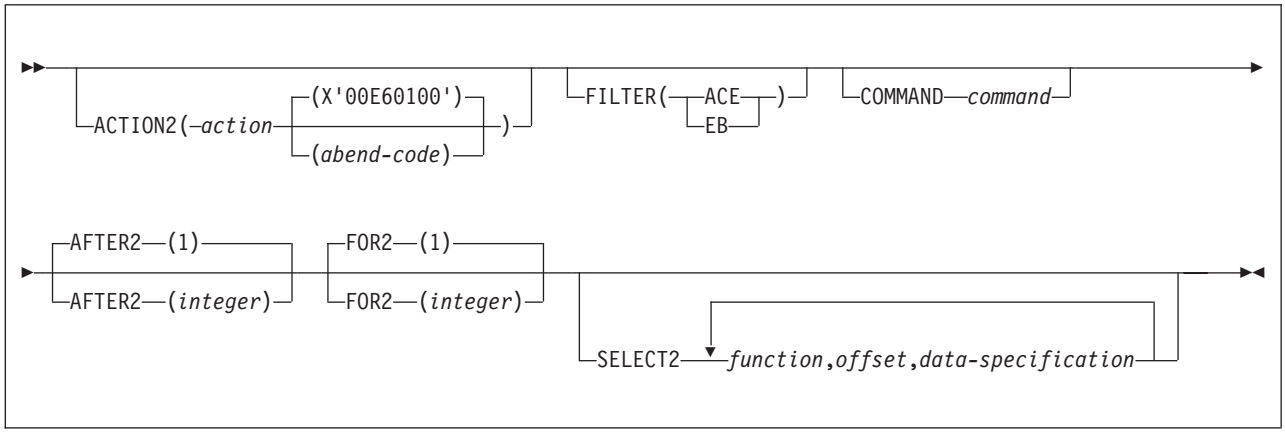
DSN1SDMP syntax diagram



Notes:

1 The options in the *second-trace-spec* do not have to be specified immediately following the STTRACE option. However, they can be specified only if the STTRACE option is also specified.

second-trace-spec:



Option descriptions

START TRACE (*trace-parameters***)**

Indicates the start of a DSN1SDMP job. START TRACE is a required keyword and must be the first keyword that is specified in the SDMPIN input stream. The trace parameters that you use are those that are described in Chapter 2 of *DB2 Command Reference*, except that you cannot use the subsystem recognition character.

If the START TRACE command in the SDMPIN input stream is not valid, or if the user is not properly authorized, the IFI (instrumentation facility interface) returns an error code and START TRACE does not take effect. DSN1SDMP writes the error message to the SDMPPRNT data set.

Trace Destination: If DB2 trace data is to be written to the SDMPTRAC data set, the trace destination must be an IFI online performance (OP) buffer. OP buffer destinations are specified in the DEST keyword of START TRACE. Eight OP buffer destinations exist, OP1 to OP8. The OPX trace destination assigns the next available OP buffer.

The DB2 output text from the START TRACE command is written to SDMPPRNT.

START TRACE and its associated keywords must be specified first. Specify the remaining selective dump keywords in any order following the START TRACE command.

SELECT *function,offset,data-specification*

Specifies selection criteria in addition to those that are specified on the START TRACE command. SELECT expands the data that is available for selection in a trace record and allows more specific selection of data in the trace record than using START TRACE alone. You can specify a maximum of eight SELECT criteria.

The selection criteria use the concept of the current-record pointer. DB2 initializes the current-record pointer to zero, that is, at the beginning of the trace record. For this instance of the DSN1SDMP trace, the trace record begins with the self-defining section. The current-record pointer can be modified by P*x* and LN functions, which are described in the list of functions below.

For information about the fields in the DB2 trace records, see Appendix D (Volume 2) of *DB2 Administration Guide*.

You can specify the selection criteria with the following parameters:

function Specifies the type of search that is to be performed on the trace record. The specified value must be two characters. The possible values are:

DR Specifies a direct comparison of data from the specified offset. The offset is always calculated from the current-record pointer.

GE Specifies a comparison of data that is greater than or equal to the value of the specified offset. The offset is always calculated from the current-record pointer. The test succeeds if the data from the specified offset is greater than or equal to *data-specification*, which you can specify on the SELECT option.

LE Specifies a comparison of data that is less than or equal to the value of the specified offset. The offset is always calculated from the current-record pointer. The test succeeds if the data from the specified offset is less than or equal to *data-specification*, which you specify on the SELECT option.

P1, P2, or P4
Selects the 1-, 2-, or 4-byte field that is located *offset* bytes past the start of the record. The function then moves the current-record pointer that number of bytes into the record. P1, P2, and P4 always start from the beginning of the record (plus the offset that you specify).

This offset is saved as the current-record pointer that is to be used on subsequent DR, LE, GR, and LN requests.

For example, suppose that the user knows that the offset to the standard header is 4 bytes long and is located in the first 4 bytes of the record. P4,00 reads that offset and moves the current-record pointer to the start of the standard header.

LN Advances the current-record pointer by the number of bytes that are indicated in the 2-byte field that is located *offset* bytes from the previous current-record pointer.

This offset is saved as the current-record pointer that is to be used on subsequent DR, LE, GR, and LN requests.

offset Specifies the number (in decimal) of bytes into the trace record where the comparison with the *data-specification* field begins. The offset starts from the beginning of the trace record after a P1, P2, or P4, and from the current-record pointer after a GE, LE, LN, or DR.

The format of the DB2 trace record at *data-specification* comparison time is shown in Figure 144.

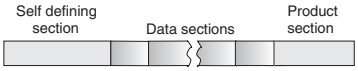


Figure 144. Format of the DB2 trace record at data specification comparison time

- The format of the self-defining section depends on the trace type.
- The format and content of the data sections depend on the IFCID that is being recorded. Each record can have one or more data sections. Each data section can have multiple repeating groups.
- The format and content of the trace header section depends on the trace type.

For more information about the format of DB2 trace records, refer to Appendix D (Volume 2) of *DB2 Administration Guide*.

data-specification
Specifies that the data can be hexadecimal (for example, X'9FECBA10') or character (C'FIELD').

ACTION
Specifies the action to perform when a trace record passes the selection criteria of the START TRACE and SELECT keywords.

Attention: The purpose of the ACTION keyword is to facilitate problem analysis. You should use it with extreme caution because you might damage existing data. Not all abends are recoverable, even if the ABENDRET parameter is specified. Some abends might force the DB2 subsystem to terminate, particularly those abends that occur during end-of-task or end-of-memory processing due to the agent having experienced a previous abend.

*action***(***abend-code***)**
Specifies a particular action to perform. Possible values for *action* are:

ABENDRET ABEND and retry the agent.

ABENDTER ABEND and terminate the agent.

An abend reason code can also be specified on this parameter. The codes must be in the range X'00E60100' to X'00E60199'. The default value is X'00E60100'.

STTRACE
Specifies that a second trace is to be started when a trace record passes the selection criteria.

If you do not specify *action* or STTRACE, the record is written and no action is performed.

AFTER(*integer***)**
Specifies that the ACTION is to be performed after the trace point is reached *integer* times.

integer must be between 1 and 32767. The **default** is **AFTER(1)**.

FOR(*integer***)**
Specifies the number of times that the ACTION is to take place when the specified trace point is reached. After *integer* times, the trace is stopped, and DSN1SDMP terminates.

integer must be between 1 and 32767 and includes the first action. If no SELECT criteria are specified, use an integer greater than 1; the START TRACE command automatically causes the action to take place one time. The **default** is **FOR(1)**.

ACTION2
Specifies the action to perform when a trace record passes the selection criteria of the START TRACE, SELECT, and SELECT2 keywords.

Attention: The ACTION2 keyword, like the ACTION keyword, should be used with extreme caution, because you might damage existing data. Not all abends are recoverable, even if the ABENDRET parameter is specified. Some abends might force the DB2 subsystem to terminate, particularly those that occur during end-of-task or end-of-memory processing due to the agent having experienced a previous abend.

*action***(***abend-code***)**
Specifies a particular action to perform. Possible values for *action* are:

ABENDRET ABEND and retry the agent.

ABENDTER ABEND and terminate the agent.

An abend reason code can also be specified on this parameter. The codes must be in the range X'00E60100-00E60199'. If no abend code is specified, X'00E60100' is used.

If you do not specify *action*, the record is written and no action is performed.

FILTER
Specifies that DSN1SDMP is to filter the output of the second trace based on either an ACE or an EB.

(ACE)
Specifies that DSN1SDMP is to include trace records only for the agent control element (ACE) that is associated with the agent when the first action is triggered and the second trace is started.

(EB)
Specifies that DSN1SDMP is to include trace records only for the execution block (EB) that is associated with the agent when the first action is triggered and the second trace is started.

COMMAND
Indicates that the specified command is to be issued when a trace record passes the selection criteria for the first trace and a second trace is started. You can start a second trace by specifying the STTRACE option.

command
Specifies a specific command to be issued. For a complete list of commands, see *DB2 Command Reference*.

FOR2(*integer***)**
Specifies the number of times that the ACTION2 is to take place when the specified second trace point is reached. After *integer* times, the second trace is stopped, and DSN1SDMP terminates.

integer must be between 1 and 32767 and includes the first action. If no SELECT2 criteria are specified, use an integer greater than 1; the STTRACE option automatically causes the action to take place one time. The **default** is **FOR2(1)**.

AFTER2(*integer***)**
Specifies that the ACTION2 is to be performed after the second trace point is reached *integer* times.

integer must be between 1 and 32767. The **default** is **AFTER2(1)**.

SELECT2 *function,offset,data-specification*
Specifies selection criteria for the second trace. This option functions like the SELECT option, except that it pertains to the second trace only. You can start a second trace by specifying the STTRACE option.

Before running DSN1SDMP

This section contains information that you need to know before you run DSN1SDMP.

Environment

Run DSN1SDMP as a z/OS job, and execute it with the DSN TSO command processor. To execute DSN1SDMP, the DB2 subsystem must be running.

The z/OS job completes only under one of the following conditions:

- The TRACE and any additional selection criteria that are started by DSN1SDMP meet the criteria specified in the FOR parameter.
- The TRACE that is started by DSN1SDMP is stopped by using the STOP TRACE command.
- The job is canceled by the operator.

If you must stop DSN1SDMP, use the STOP TRACE command.

Authorization required

To execute this utility, the privilege set of the process must include one of the following privileges or authorities:

- TRACE system privilege
- SYSOPR authority
- SYSADM authority
- MONITOR1 or MONITOR2 privileges (if you are using user-defined data sets)

The user who executes DSN1SDMP must have EXECUTE authority on the plan that is specified in the *trace-parameters* of the START TRACE keyword.

Control statement

See “Syntax and options of the DSN1SDMP control statement” on page 777 for DSN1SDMP syntax and option descriptions.

Required data sets: DSN1SDMP uses the following DD statements:

SDMPIN Defines the control data set that specifies the input parameters to DSN1SDMP. This DD statement is required. The LRECL is 80. Only the first 72 columns are checked by DSN1SDMP.

SDMPPRNT Defines the sequential message data set that is used for

DSN1SDMP messages. If the SDMPPRNT DD statement is omitted, no messages are written. The LRECL is 131.

SYSABEND Defines the data set that is to contain an ABEND dump in case DSN1SDMP abends. This DD statement is optional.

SDMPTRAC Defines the sequential DB2 trace record data set that DB2 returns to DSN1SDMP. The DD statement is required only if trace data is written to an OPX trace destination. If the destination is anything other than an OPX buffer, SDMPTRAC is ignored.

Trace records that DB2 writes to SDMPTRAC are of the same format as SMF or GTF records except that the SDMPTRAC trace record headers contain the monitor header (that is mapped by DSNDQWIW). The DCB parameters are VB, BLKSIZE=32760, LRECL=32756.

SYSTSIN Defines the DSN commands to connect to DB2 and to execute an IFC selective dump:

```
DSN SYSTEM(subsystem name)
RUN PROG(DSN1SDMP) LIB('prefix.SDSNLOAD') PLAN(DSNEDCL)
```

The DB2 subsystem name must be filled in by the user. The DSN RUN command must specify a plan for which the user has execute authority. DSN1SDMP dump does not execute the specified plan; the plan is used only to connect to DB2.

When no plan name is specified on the DSN RUN command, the default plan name is the program name. When DSN1SDMP is executed without a plan, DSN generates an error if no DSN1SDMP plan exists for which the user has execute authority.

Using DSN1SDMP to force dumps and write trace records

This section describes the following tasks that are associated with running the DSN1SDMP utility:

"Assigning buffers"
"Conditions for generating a dump" on page 784
"Stopping or modifying DSN1SDMP traces" on page 784

Assigning buffers

The OPX trace destination assigns the next available OP buffer. You must specify the OPX destination for all traces that are being recorded to an OP*n* buffer, thereby avoiding the possibility of starting a trace to a buffer that has already been assigned.

If a trace is started to an OP*n* buffer that has already been assigned, DSN1SDMP waits indefinitely until the trace is manually stopped. The default for MONITOR-type traces is the OPX destination (the next available OP buffer). Other trace types must be explicitly directed to OP destinations via the DEST keyword of the START TRACE command. DSN1SDMP interrogates the IFCAOPN field after the START TRACE COMMAND call to determine if the trace was started to an OP buffer.

Trace records are written to the SDMPTRAC data set when the trace destination is an OP buffer (see "Trace Destination" on page 778). The instrumentation facilities

component (IFC) writes trace records to the buffer and posts DSN1SDMP to read the buffer when it fills to half of the buffer size.

You can specify the buffer size on the BUFSIZE keyword of the START TRACE command. All returned records are written to SDMPTRAC.

If the number of generated trace records requires a larger buffer size than was specified, you can lose some trace records. If this happens, error message DSN2724I is issued.

Conditions for generating a dump

DSN1SDMP generates a DB2 dump when all of the following events occur:
- DB2 produces a trace record that satisfies all of the selection criteria.
- You specify an abend action (ABENDRET or ABENDTER).
- The AFTER and FOR conditions for the trace are satisfied.

If all three events occur, an 00E601*xx* abend occurs. *xx* is an integer between 1 and 99 that DB2 obtains from the user-specified value on the ACTION keyword.

Stopping or modifying DSN1SDMP traces

If you must stop DSN1SDMP, use the STOP TRACE command.

If DSN1SDMP does not finish execution, you can stop the utility by issuing the STOP TRACE command, as in the following example:

```
-STOP TRACE=P CLASS(32)
```

DSN1SDMP executes as a stand-alone batch utility without requiring external intervention from the console operator or other programs. During execution, DSN1SDMP issues an IFI READA request to obtain the data from the OP*n* buffer and a STOP TRACE command to terminate the original trace that is started by DSN1SDMP.

A STOP TRACE or MODIFY TRACE command that is entered from a console for the trace that is started by DSN1SDMP causes immediate abnormal termination of DSN1SDMP processing. The IFI READA function terminates with an appropriate IFI termination message and reason code. Additional error messages and reason codes that are associated with the DSN1SDMP STOP TRACE command vary depending on the specific trace command that is entered by the console operator.

If the console operator terminates the original trace by using the STOP TRACE command, the subsequent STOP TRACE command that is issued by DSN1SDMP fails.

If the console operator enters a MODIFY TRACE command and processing of this command completes before the STOP TRACE command is issued by DSN1SDMP, the modified trace is also terminated.

Sample DSN1SDMP control statements

Example 1: Creating the JCL for DSN1SDMP. The example in Figure 145 on page 785 shows the skeleton JCL for a DSN1SDMP job.

```
//DSN1J018 JOB 'IFC SD',CLASS=A,
//         MSGLEVEL=(1,1),USER=SYSADM,PASSWORD=SYSADM,REGION=1024K
//*********************************************************************
//*
//*       THIS IS A SKELETON OF THE JCL USED TO RUN DSN1SDMP.
//*       YOU MUST INSERT SDMPIN DD.
//*
//*********************************************************************
//IFCSD    EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//STEPLIB  DD DISP=SHR,DSN=prefix.SDSNLOAD
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SDMPPRNT DD SYSOUT=*
//SDMPTRAC DD DISP=(NEW,CATLG,CATLG),DSN=IFCSD.TRACE,
//            UNIT=SYSDA,SPACE=(8192,(100,100)),DCB=(DSORG=PS,
#           LRECL=32756,RECFM=VB,BLKSIZE=32760)
#           //SDMPIN   DD *
//*********************************************************************
//*
//*       INSERT SDMPIN DD HERE.  IT MUST BEGIN WITH A VALID
//*       START TRACE COMMAND (WITHOUT THE SUBSYSTEM RECOGNITION CHAR)
//*
//*********************************************************************

                  (VALID SDMPIN GOES HERE)

/*
//*********************************************************************
//SYSUDUMP DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
 RUN PROG(DSN1SDMP) PLAN(DSNEDCL)
 END
//*
```

Figure 145. Skeleton JCL for DSN1SDMP

Example 2: Abending and terminating agent on -904 SQL CODE. The example in Figure 146 on page 786 specifies that DB2 is to start a performance trace (which is indicated by the letter P) and activate only IFCID 58. To start only those IFCIDs that are specified in the IFCID option, use trace classes 30-32. In this example, trace class 32 is specified. The trace output is to be recorded in a generic destination that uses the first free OP*n* slot, as indicated by the DEST option. These START TRACE options are explained in greater detail in *DB2 Command Reference*.

The SELECT option indicates additional criteria for data in the trace record. In this example, the SELECT option first specifies that the current-record pointer is to be moved to the 4-byte field that is located 8 bytes past the start of the record. The option then specifies that the utility is to directly compare the data that is 74 bytes from the current-record pointer with the value X'FFFFFC78'.

When a trace record passes the selection criteria of the START TRACE command and SELECT keywords, DSN1SDMP is to perform the action that is specified by the ACTION keyword. In this example, the job is to abend and terminate with reason code 00E60188. This action is to take place only once, as indicated by the FOR option. FOR(1) is the default, and is therefore not required to be explicitly specified.

```
//SDMPIN   DD *
* START ONLY IFCID 58, END SQL STATEMENT
 START TRACE=P CLASS(32) IFCID(58) DEST(OPX)
 FOR(1)
 ACTION(ABENDTER(00E60188))
 SELECT
* OFFSET TO FIRST DATA SECTION CONTAINING THE SQLCA.
 P4,08
* SQLCODE -904, RESOURCE UNAVAILABLE
 DR,74,X'FFFFFC78'
/*
```

Figure 146. Example job that abends and terminates agent on -904 SQL code

Example 3: Abending and retrying on RMID 20. The example in Figure 147 specifies that DB2 is to start a performance trace (which is indicated by the letter P) and activate all IFCIDs in classes 3 and 8. The trace output is to be recorded in a generic destination that uses the first free OP*n* slot, as indicated by the DEST option. The TDATA (TRA) option specifies that a CPU header is to be placed into the product section of each trace record. These START TRACE options are explained in greater detail in *DB2 Command Reference*.

The SELECT option indicates additional criteria for data in the trace record. In this example, the SELECT option first specifies that the current-record pointer is to be placed at the 4-byte field that is located at the start of the record. The current record pointer is then to be advanced the number of bytes that are indicated in the 2-byte field that is located at the current record pointer. The utility is then to directly compare the data that is 4 bytes from the current-record pointer with the value X'0025'.

When a trace record passes the selection criteria of the START TRACE command and SELECT keywords, DSN1SDMP is to perform the action that is specified by the ACTION keyword. In this example, the job is to abend and retry the agent.

```
//*       ABEND AND RETRY AN AGENT WHEN EVENT ID X'0025'
//*       (AGENT ALLOCATION) IS RECORDED BY RMID 20 (SERVICE
//*       CONTROLLER).
//*
//SDMPIN   DD *
*  ENSURE ONLY THE TRACE HEADER IS APPENDED WITH THE STANDARD HEADER
*  VIA THE TDATA KEYWORD ON START TRACE
 START TRACE=P CLASS(3,8) RMID(20) DEST(OPX) TDATA(TRA)
*  ABEND AND RETRY THE AGENT WITH THE DEFAULT ABEND CODE (00E60100)
 ACTION(ABENDRET)
*  SPECIFY THE SELECT CRITERIA FOR RMID.EID
 SELECT
*  OFFSET TO THE STANDARD HEADER
 P4,00
*  ADD LENGTH OF STANDARD HEADER TO GET TO TRACE HEADER
 LN,00
*  LOOK FOR EID 37 AT OFFSET 4 IN THE TRACE HEADER
 DR,04,X'0025'
/*
```

Figure 147. Example job that abends and retries on RMID 20

Example 4: Generating a dump on SQLCODE -811 RMID16 IFCID 58. The example in Figure 148 on page 787 specifies that DB2 is to start a performance trace (which is indicated by the letter P) and activate all IFCIDs in class 3. The trace output is to be recorded in the system management facility (SMF). The TDATA (COR,TRA)

option specifies that a trace header and a CPU header are to be placed into the product section of each trace record. These START TRACE options are explained in greater detail in *DB2 Command Reference*.

The SELECT option indicates additional criteria for data in the trace record. In this example, the SELECT option first specifies that the current-record pointer is to be placed at the 4-byte field that is located at the start of the record. The utility is then to directly compare the data that is 2 bytes from the current-record pointer with the value X'0116003A'. The current record pointer is then to be moved to the 4-byte field that is located 8 bytes past the start of the current record. The utility is then to directly compare the data that is 74 bytes from the current-record pointer with the value X'FFFFFCD5'.

When a trace record passes the selection criteria of the START TRACE command and SELECT keywords, DSN1SDMP is to perform the action that is specified by the ACTION keyword. In this example, the job is to abend with reason code 00E60188 and retry the agent. This action is to take place only once, as indicated by the FOR option. FOR(1) is the default, and is therefore not required to be explicitly specified. AFTER(1) indicates that this action is to be performed the first time the trace point is reached. AFTER(1) is also the default.

```
//SDMPIN   DD *
 START TRACE=P CLASS(3) RMID(22) DEST(SMF) TDATA(COR,TRA)
 AFTER(1)
 FOR(1)
 SELECT
* POSITION TO HEADERS (QWHS IS ALWAYS FIRST)
  P4,00
* CHECK QWHS 01, FOR RMID 16, IFCID 58
  DR,02,X'0116003A'
* POSITION TO SECOND SECTION (1ST DATA SECTION)
  P4,08
* COMPARE SQLCODE FOR 811
  DR,74,X'FFFFFCD5'
 ACTION(ABENDRET(00E60188))
/*
```

Figure 148. Example job that generates a dump on SQL code -811 RMID16 IFCID

Example 5: Starting a second trace. The example job in Figure 149 on page 788 starts a trace on IFC 196 records. An IFC 196 record is written when a lock timeout occurs. In this example, when a lock timeout occurs, DSN1SDMP is to start a second trace, as indicated by the ACTION(STTRACE) option. This second trace is to be an accounting trace, as indicated by the COMMAND START TRACE(ACCTG) option. This trace is to include records only for the ACE that is associated with the agent that timed out, as indicated by the FILTER(ACE) option. When the qualifying accounting record is found, DSN1SDMP generates a dump.

```
//SDMPIN   DD   *
* START ONLY IFCID 196, TIMEOUT
 START TRACE=P CLASS(32) IFCID(196) DEST(SMF)
 AFTER(1)
* ACTION = START ACCOUNTING TRACE
 ACTION(STTRACE)
* FILTER ON JUST 196 RECORDS...
 SELECT
 P4,00
 DR,04,X'00C4'
* WHEN ACCOUNTING IS CUT, ABEND
 ACTION2(ABENDRET(00E60188))
* START THE ACCOUNTING TRACE FILTER ON THE ACE OF THE AGENT
* THAT TIMED OUT
 COMMAND
 START TRACE(ACCTG) CLASS(32) IFCID(3) DEST(SMF)
* Filter can be for ACE or EB
 FILTER(ACE)
 /*
```

Figure 149. Example job that starts a second trace.

DSN1SDMP output

One intended use of this utility is to aid in determining and correcting system problems. When diagnosing DB2, you might need to refer to licensed documentation to interpret output from this utility. For more information about diagnosing problems, see *DB2 Diagnosis Guide and Reference*.

Part 4. Appendixes

Appendix A. Limits in DB2 UDB for z/OS

System storage limits might preclude the limits specified in this section. The limit for items not that are not specified below is limited by system storage.

Table 143 shows the length limits for identifiers.

Table 143. Identifier length limits. The term *byte* in this table means the number of bytes for the UTF-8 representation unless noted otherwise.

Item	Limit
External-java-routine-name	1305 bytes
Name of an alias, auxiliary table, collection, constraint, correlation, cursor (except for DECLARE CURSOR WITH RETURN or the EXEC SQL utility), distinct type (both parts of two-part name), function (both parts of two-part name), host identifier, index, JARs, parameter, procedure, schema, sequence, specific, statement, storage group, savepoint, SQL condition, SQL label, SQL parameter, SQL variable, synonym, table, trigger, view, XML attribute name, XML element name	128 bytes
Name of an authorization ID	8 bytes
# Version identifier	64 bytes
Name of a column	30 bytes
Name of cursor that is created with DECLARE CURSOR WITH RETURN	30 bytes
Name of cursor that is created with the EXEC SQL utility	8 bytes
Name of a location	16 bytes
Name of buffer pool name, catalog, database, plan, program, table space	8 bytes
Name of package	8 bytes (Only 8 EBCDIC characters are used for packages that are created with the BIND PACKAGE command. 128 bytes can be used for packages that are created as a result of the CREATE TRIGGER statement.)

Table 144 shows the minimum and maximum limits for numeric values.

Table 144. Numeric limits

Item	Limit
Smallest SMALLINT value	-32768
Largest SMALLINT value	32767
Smallest INTEGER value	-2147483648
Largest INTEGER value	2147483647
Smallest REAL value	About -7.2×10^{75}
Largest REAL value	About 7.2×10^{75}
Smallest positive REAL value	About 5.4×10^{-79}
Largest negative REAL value	About -5.4×10^{-79}

Table 144. Numeric limits (continued)

Item	Limit
Smallest FLOAT value	About -7.2×10^{75}
Largest FLOAT value	About 7.2×10^{75}
Smallest positive FLOAT value	About 5.4×10^{-79}
Largest negative FLOAT value	About -5.4×10^{-79}
Smallest DECIMAL value	$1 - 10^{31}$
Largest DECIMAL value	$10^{31} - 1$
Largest decimal precision	31

Table 145 shows the length limits for strings.

Table 145. String length limits

Item	Limit
Maximum length of CHAR	255 bytes
Maximum length of GRAPHIC	127 double-byte characters
Maximum length[1] of VARCHAR	4046 bytes for 4-KB pages 8128 bytes for 8-KB pages 16320 bytes for 16-KB pages 32704 bytes for 32-KB pages
Maximum length[1] of VARGRAPHIC	2023 double-byte characters for 4-KB pages 4064 double-byte characters for 8-KB pages 8160 double-byte characters for 16-KB pages 16352 double-byte characters for 32-KB pages
Maximum length of CLOB	2 147 483 647 bytes (2 GB - 1 byte)
Maximum length of DBCLOB	1 073 741 823 double-byte characters
Maximum length of BLOB	2 147 483 647 bytes (2 GB - 1 byte)
Maximum length of a character constant	32704 UTF-8 bytes
Maximum length of a hexadecimal character constant	32704 hexadecimal digits
Maximum length of a graphic string constant	16352 double-byte characters (32704 bytes when expressed in UTF-8)
Maximum length of a hexadecimal graphic string constant	32704 hexadecimal digits
Maximum length of a concatenated character string	2 147 483 647 bytes (2 GB - 1 byte)
Maximum length of a concatenated graphic string	1 073 741 824 double-byte characters
Maximum length of a concatenated binary string	2 147 483 647 bytes (2 GB - 1 byte)

Note:

1. The maximum length can be achieved only if the column is the only column in the table. Otherwise, the maximum length depends on the amount of space remaining on a page.

Table 146 shows the minimum and maximum limits for datetime values.

Table 146. Datetime limits

Item	Limit
Smallest DATE value (shown in ISO format)	0001-01-01

Table 146. Datetime limits (continued)

Item	Limit
Largest DATE value (shown in ISO format)	9999-12-31
Smallest TIME value (shown in ISO format)	00.00.00
Largest TIME value (shown in ISO format)	24.00.00
Smallest TIMESTAMP value	0001-01-01-00.00.00.000000
Largest TIMESTAMP value	9999-12-31-24.00.00.000000

Table 147 shows the DB2 limits on SQL statements.

Table 147. DB2 limits on SQL statements

Item	Limit
Maximum number of columns that are in a table or view (the value depends on the complexity of the CREATE VIEW statement) or columns returned by a table function.	750 or fewer 749 if the table is a dependent
Maximum number of base tables in a view, SELECT, UPDATE, INSERT, or DELETE	225
Maximum row and record sizes for a table	See the description of CREATE TABLE in Chapter 5 of *DB2 SQL Reference*.
Maximum number of volume IDs in a storage group	133
Maximum number of partitions in a partitioned table space or partitioned index	64 for table spaces that are not defined with LARGE or a DSSIZE greater than 2 GB. 4096, depending on what is specified for DDSIZE or LARGE and the page size. See the description of NUMPARTS in CREATE TABLESPACE in Chapter 5 of *DB2 SQL Reference*.
Maximum sum of the lengths of limit key values of a partition boundary	765 UTF-8 bytes
Maximum size of a partition (table space or index)	For table spaces that are not defined with LARGE or a DSSIZE greater than 2 GB: 4 GB, for 1 to 16 partitions 2 GB, for 17 to 32 partitions 1 GB, for 33 to 64 partitions For table spaces that are defined with LARGE: 4 GB, for 1 to 4096 partitions For table spaces that are defined with a DSSIZE greater than 2 GB: 64 GB, depending on the page size (for 1 to 256 partitions for 4-KB pages, for 1 to 512 partitions for 8-KB pages, for 1 to 1024 partitions for 16-KB pages, and 1 to 2048 partitions for 32-KB pages)
Maximum size of a DBRM entry	3272000 bytes
Maximum length of an index key	Partitioning index: 255-*n* Nonpartitioning index that is padded: 2000-*n* Nonpartitioning index that is not padded: 2000-*n*-2m Where *n* is the number of columns in the key that allow nulls and *m* is the number of varying-length columns in the key

Table 147. DB2 limits on SQL statements (continued)

Item	Limit
Maximum number of bytes used in the partitioning of a partitioned index	255 (This maximum limit is subject to additional limitations, depending on the number of partitions in the table space. The number of partitions * (106 + limit key size) must be less than 65394.)
Maximum number of columns in an index key	64
Maximum number of tables in a FROM clause	225 or fewer, depending on the complexity of the statement
Maximum number of subqueries in a statement	224
Maximum total length of host and indicator variables pointed to in an SQLDA	32767 bytes 2 147 483 647 bytes (2 GB - 1 byte) for a LOB, subject to the limitations that are imposed by the application environment and host language
Longest host variable used for insert or update	32704 bytes for a non-LOB 2 147 483 647 bytes (2 GB - 1 byte) for a LOB, subject to the limitations that are imposed by the application environment and host language
Longest SQL statement	2 097 152 bytes
Maximum number of elements in a select list	750 or fewer, depending on whether the select list includes a hidden ROWID column or is for the result table of static scrollable cursor[1]
Maximum number of predicates in a WHERE or HAVING clause	Limited by storage
Maximum total length of columns of a query operation requiring a sort key (SELECT DISTINCT, ORDER BY, GROUP BY, UNION without the ALL keyword, and the DISTINCT keyword for aggregate functions)	4000 bytes
Maximum total length os columns of a query operation requiring sort and evaluating column functions (MULTIPLE DISTINCT)	32686 bytes
Maximum length of a sort key	16000 bytes
Maximum length of a check constraint	3800 bytes
Maximum number of bytes that can be passed in a single parameter of an SQL CALL statement	32765 bytes for a non-LOB 2 147 483 647 bytes (2 GB - 1 byte) for a LOB, subject to the limitations imposed by the application environment and host language
Maximum number of stored procedures, triggers, and user-defined functions that an SQL statement can implicitly or explicitly reference	16 nesting levels
Maximum length of the SQL path	2048 bytes

Note:

1. If the scrollable cursor is read-only, the maximum number is 749 less the number of columns in the ORDER BY that are not in the select list. If the scrollable cursor is not read-only, the maximum number is 747.

Table 148 on page 795 shows the DB2 system limits.

Table 148. DB2 system limits

Item	Limit
Maximum number of concurrent DB2 or application agents	Limited by the EDM pool size, buffer pool size, and the amount of storage that is used by each DB2 or application agent
Largest table or table space	128 terabytes (TB)
Largest simple or segmented table space	64 GB
Largest log space	2^{48}
Largest active log data set	4 GB -1 byte
Largest archive log data set	4 GB -1 byte
Maximum number of active log copies	2
Maximum number of archive log copies	2
Maximum number of active log data sets (each copy)	93
Maximum number of archive log volumes (each copy)	10000
Maximum number of databases accessible to an application or end user	Limited by system storage and EDM pool size
Largest EDM pool	The installation parameter maximum depends on available space
Maximum number of databases	65271
Maximum number of rows per page	255 for all table spaces except catalog and directory tables spaces, which have a maximum of 127
Maximum simple or segmented data set size	2 GB
Maximum partitioned data set size	See item “maximum size of a partition” in Table 147 on page 793
Maximum LOB data set size	64 GB
Maximum number of rows that can be inserted with a single INSERT statement	32767 rows

Appendix B. DB2-supplied stored procedures

DB2 provides several stored procedures that you can call in your application programs to perform a number of utility and application programming functions.

Note: These stored procedures do not propagate the transaction identifier (XID) of the thread. These stored procedures run under a new private context rather than under the native context of the task that called it.

DB2 provides the following stored procedures:

- The utilities stored procedure for EBCDIC input (DSNUTILS)

  This stored procedure lets you invoke utilities from a local or remote client program. See "Invoking utilities as a stored procedure (DSNUTILS)" on page 799 for information.

- The utilities stored procedure for Unicode input (DSNUTILU)

  This stored procedure lets you invoke utilities from a local or remote client program that generates Unicode utility control statements. See "DSNUTILU stored procedure" on page 809 for information.

- The DB2 Universal Database Control Center (Control Center) table space and index information stored procedure (DSNACCQC)

  This stored procedure helps you determine when utilities should be run on your databases. This stored procedure is designed primarily for use by the Control Center but can be invoked from any client program. See "The Control Center table space and index information stored procedure (DSNACCQC)" on page 812 for information.

- The Control Center partition information stored procedure (DSNACCAV)

  This stored procedure helps you determine when utilities should be run on your partitioned table spaces. This stored procedure is designed primarily for use by the Control Center but can be invoked from any client program. See "The Control Center partition information stored procedure (DSNACCAV)" on page 820 for information.

- The real-time statistics stored procedure (DSNACCOR)

  This stored procedure queries the DB2 real-time statistics tables to help you determine when you should run COPY, REORG, or RUNSTATS, or enlarge your DB2 data sets. See "The DB2 real-time statistics stored procedure" on page 830 for more information.

- The WLM environment refresh stored procedure (WLM_REFRESH)

  This stored procedure lets you refresh a WLM environment from a remote workstation. See Appendix I of *DB2 Application Programming and SQL Guide* for information.

- The CICS transaction invocation stored procedure (DSNACICS)

  This stored procedure lets you invoke CICS transactions from a remote workstation. See Appendix I of *DB2 Application Programming and SQL Guide* for more information.

- The SYSIBM.USERNAMES encryption stored procedure (DSNLEUSR)

  This stored procedure lets you store encrypted values in the NEWAUTHID and PASSWORD fields of the SYSIBM.USERNAMES catalog table. See Appendix I of *DB2 Administration Guide* for more information.

- The IMS transactions stored procedure (DSNAIMS)

This stored procedure allows DB2 to invoke IMS transactions and commands easily, without maintaining their own connections to IMS.

- The IMS transactions stored procedure with multi-segment input support (DSNAIMS2)

  This stored procedure offers the same functionality as the DSNAIMS stored procedure with the addition of multi-segment input support for IMS transactions.

- The EXPLAIN stored procedure (DSN8EXP)

  This stored procedure allows a user to perform an EXPLAIN on an SQL statement without having the authorization to execute that SQL statement.

- The MQ XML stored procedures

  All of the MQ XML stored procedures have been deprecated.

  These stored procedures perform the following functions:

Table 149. MQ XML stored procedures

Stored procedure name	Function	For information, see:
DXXMQINSERT	Returns a message that contains an XML document from an MQ message queue, decomposes the document, and stores the data in DB2 tables that are specified by an enabled XML collection.	*DB2 Application Programming and SQL Guide*
DXXMQSHRED	Returns a message that contains an XML document from an MQ message queue, decomposes the document, and stores the data in DB2 tables that are specified in a document access definition (DAD) file. DXXMQSHRED does not require an enabled XML collection.	*DB2 Application Programming and SQL Guide*
DXXMQINSERTCLOB	Returns a message that contains an XML document from an MQ message queue, decomposes the document, and stores the data in DB2 tables that are specified by an enabled XML collection. DXXMQINSERTCLOB is intended for an XML document with a length of up to 1MB.	*DB2 Application Programming and SQL Guide*
DXXMQSHREDCLOB	Returns a message that contains an XML document from an MQ message queue, decomposes the document, and stores the data in DB2 tables that are specified in a document access definition (DAD) file. DXXMQSHREDCLOB does not require an enabled XML collection. DXXMQSHREDCLOB is intended for an XML document with a length of up to 1MB.	*DB2 Application Programming and SQL Guide*
DXXMQINSERTALL	Returns messages that contains XML documents from an MQ message queue, decomposes the documents, and stores the data in DB2 tables that are specified by an enabled XML collection. DXXMQINSERTALL is intended for XML documents with a length of up to 3KB.	*DB2 Application Programming and SQL Guide*

Table 149. MQ XML stored procedures (continued)

Stored procedure name	Function	For information, see:
DXXMQSHREDALL	Returns messages that contain XML documents from an MQ message queue, decomposes the documents, and stores the data in DB2 tables that are specified in a document access definition (DAD) file. DXXMQSHREDALL does not require an enabled XML collection. DXXMQSHREDALL is intended for XML documents with a length of up to 3KB.	*DB2 Application Programming and SQL Guide*
DXXMQSHREDALLCLOB	Returns messages that contain XML documents from an MQ message queue, decomposes the documents, and stores the data in DB2 tables that are specified in a document access definition (DAD) file. DXXMQSHREDALLCLOB does not require an enabled XML collection. DXXMQSHREDALLCLOB is intended for XML documents with a length of up to 1MB.	*DB2 Application Programming and SQL Guide*
DXXMQINSERTALLCLOB	Returns messages that contains XML documents from an MQ message queue, decomposes the documents, and stores the data in DB2 tables that are specified by an enabled XML collection. DXXMQINSERTALLCLOB is intended for XML documents with a length of up to 1MB.	*DB2 Application Programming and SQL Guide*
DXXMQGEN	Constructs XML documents from data that is stored in DB2 tables that are specified in a document access definition (DAD) file, and sends the XML documents to an MQ message queue. DXXMQGEN is intended for XML documents with a length of up to 3KB.	*DB2 Application Programming and SQL Guide*
DXXMQRETRIEVE	Constructs XML documents from data that is stored in DB2 tables that are specified in an enabled XML collection, and sends the XML documents to an MQ message queue. DXXMQRETRIEVE is intended for XML documents with a length of up to 3KB.	*DB2 Application Programming and SQL Guide*
DXXMQGENCLOB	Constructs XML documents from data that is stored in DB2 tables that are specified in a document access definition (DAD) file, and sends the XML documents to an MQ message queue. DXXMQGENCLOB is intended for XML documents with a length of up to 32KB.	*DB2 Application Programming and SQL Guide*
DXXMQRETRIEVECLOB	Constructs XML documents from data that is stored in DB2 tables that are specified in an enabled XML collection, and sends the XML documents to an MQ message queue. DXXMQRETRIEVECLOB is intended for XML documents with a length of up to 32KB.	*DB2 Application Programming and SQL Guide*

Invoking utilities as a stored procedure (DSNUTILS)

The DSNUTILS stored procedure enables you use the SQL CALL statement to execute DB2 utilities from a DB2 application program that specifies EBCDIC input. When called, DSNUTILS performs the following actions:

- Dynamically allocates the specified data sets

- Creates the utility input (SYSIN) stream
- Invokes DB2 utilities (program DSNUTILB)
- Deletes all the rows that are currently in the created temporary table (SYSIBM.SYSPRINT)
- Captures the utility output stream (SYSPRINT) into a created temporary table (SYSIBM.SYSPRINT)
- Declares a cursor to select from SYSPRINT:

```
DECLARE SYSPRINT CURSOR WITH RETURN FOR
  SELECT SEQNO, TEXT FROM SYSPRINT
  ORDER BY SEQNO;
```

- Opens the SYSPRINT cursor and returns.

The calling program then fetches from the returned result set to obtain the captured utility output.

Environment for DSNUTILS

DSNUTILS **must** run in a WLM environment. TCB=1 is also required.

Authorization required for DSNUTILS

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNUTILS
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

Then, to execute the utility, you must use a privilege set that includes the authorization to run the specified utility.

Control statement for DSNUTILS

DSNUTILS dynamically allocates the specified data sets. Any utility that requires a sort must include the SORTDEVT keyword in the utility control statement, and optionally, the SORTNUM keyword.

If the DSNUTILS stored procedure invokes a new utility, refer to Table 150 for information about the default data dispositions that are specified for dynamically allocated data sets. This table lists the DD name that is used to identify the data set and the default dispositions for the data set by utility.

Table 150. Data dispositions for dynamically allocated data sets

DD name	CHECK DATA	CHECK INDEX or CHECK LOB	COPY	COPY-TOCOPY	LOAD	MERGE-COPY	REBUILD INDEX	REORG INDEX	REORG TABLE-SPACE	UNLOAD
SYSREC	ignored	ignored	ignored	ignored	OLD KEEP KEEP	ignored	ignored	ignored	NEW CATLG CATLG	NEW CATLG CATLG
SYSDISC	ignored	ignored	ignored	ignored	NEW CATLG CATLG	ignored	ignored	ignored	NEW CATLG CATLG	ignored
SYSPUNCH	ignored	ignored	ignored	ignored	ignored	ignored	ignored	ignored	NEW CATLG CATLG	NEW CATLG CATLG
SYSCOPY	ignored	ignored	NEW CATLG CATLG	ignored	NEW CATLG CATLG	NEW CATLG CATLG	ignored	ignored	NEW CATLG CATLG	ignored

Table 150. Data dispositions for dynamically allocated data sets (continued)

DD name	CHECK DATA	CHECK INDEX or CHECK LOB	COPY	COPY-TOCOPY	LOAD	MERGE-COPY	REBUILD INDEX	REORG INDEX	REORG TABLE-SPACE	UNLOAD
SYSCOPY2	ignored	ignored	NEW CATLG CATLG	NEW CATLG CATLG	NEW CATLG CATLG	NEW CATLG CATLG	ignored	ignored	NEW CATLG CATLG	ignored
SYSRCPY1	ignored	ignored	NEW CATLG CATLG	NEW CATLG CATLG	NEW CATLG CATLG	NEW CATLG CATLG	ignored	ignored	NEW CATLG CATLG	ignored
SYSRCPY2	ignored	ignored	NEW CATLG CATLG	NEW CATLG CATLG	NEW CATLG CATLG	NEW CATLG CATLG	ignored	ignored	NEW CATLG CATLG	ignored
SYSUT1	NEW DELETE CATLG	NEW DELETE CATLG	ignored	ignored	NEW DELETE CATLG	ignored	NEW DELETE CATLG	NEW CATLG CATLG	NEW DELETE CATLG	ignored
SORTOUT	NEW DELETE CATLG	ignored	ignored	ignored	NEW DELETE CATLG	ignored	ignored	ignored	NEW DELETE CATLG	ignored
SYSMAP	ignored	ignored	ignored	ignored	NEW CATLG CATLG	ignored	ignored	ignored	ignored	ignored
SYSERR	NEW CATLG CATLG	ignored	ignored	ignored	NEW CATLG CATLG	ignored	ignored	ignored	ignored	ignored
FILTER	ignored	ignored	NEW DELETE CATLG	ignored	ignored	ignored	ignored	ignored	ignored	ignored

If the DSNUTILS stored procedure restarts a current utility, refer to Table 151 for information about the default data dispositions that are specified for dynamically-allocated data sets on RESTART. This table lists the DD name that is used to identify the data set and the default dispositions for the data set by utility.

Table 151. Data dispositions for dynamically allocated data sets on RESTART

DD name	CHECK DATA	CHECK INDEX or CHECK LOB	COPY	COPY-TOCOPY	LOAD	MERGE-COPY	REBUILD INDEX	REORG INDEX	REORG TABLE-SPACE	UNLOAD
SYSREC	ignored	ignored	ignored	ignored	OLD KEEP KEEP	ignored	ignored	ignored	MOD CATLG CATLG	MOD CATLG CATLG
SYSDISC	ignored	ignored	ignored	ignored	MOD CATLG CATLG	ignored	ignored	ignored	MOD CATLG CATLG	ignored
SYSPUNCH	ignored	ignored	ignored	ignored	ignored	ignored	ignored	ignored	MOD CATLG CATLG	MOD CATLG CATLG
SYSCOPY	ignored	ignored	MOD CATLG CATLG	ignored	MOD CATLG CATLG	MOD CATLG CATLG	ignored	ignored	MOD CATLG CATLG	ignored
SYSCOPY2	ignored	ignored	MOD CATLG CATLG	MOD CATLG CATLG	MOD CATLG CATLG	MOD CATLG CATLG	ignored	ignored	MOD CATLG CATLG	ignored
SYSRCPY1	ignored	ignored	MOD CATLG CATLG	MOD CATLG CATLG	MOD CATLG CATLG	MOD CATLG CATLG	ignored	ignored	MOD CATLG CATLG	ignored
SYSRCPY2	ignored	ignored	MOD CATLG CATLG	MOD CATLG CATLG	MOD CATLG CATLG	MOD CATLG CATLG	ignored	ignored	MOD CATLG CATLG	ignored
SYSUT1	MOD DELETE CATLG	MOD DELETE CATLG	ignored	ignored	MOD DELETE CATLG	ignored	MOD DELETE CATLG	MOD CATLG CATLG	MOD DELETE CATLG	ignored

Table 151. Data dispositions for dynamically allocated data sets on RESTART (continued)

DD name	CHECK DATA	CHECK INDEX or CHECK LOB	COPY	COPY-TOCOPY	LOAD	MERGE-COPY	REBUILD INDEX	REORG INDEX	REORG TABLE-SPACE	UNLOAD
SORTOUT	MOD DELETE CATLG	ignored	ignored	ignored	MOD DELETE CATLG	ignored	ignored	ignored	MOD DELETE CATLG	ignored
SYSMAP	ignored	ignored	ignored	ignored	MOD CATLG CATLG	ignored	ignored	ignored	ignored	ignored
SYSERR	MOD CATLG CATLG	ignored	ignored	ignored	MOD CATLG CATLG	ignored	ignored	ignored	ignored	ignored
FILTER	ignored	ignored	MOD DELETE CATLG	ignored	ignored	ignored	ignored	ignored	ignored	ignored

DSNUTILS stored procedure syntax diagram

The following syntax diagram shows the SQL CALL statement for invoking utilities as a stored procedure. Because the linkage convention for DSNUTILS is GENERAL, you cannot pass null values for input parameters. For character parameters that you are not using, specify an empty string ('').

```
►►—CALL—DSNUTILS—(—utility-id,restart,utstmt,retcode—,—utility-name———————————————►
►—,recdsn,recdevt,recspace—,discdsn,discdevt,discspace—,pnchdsn,pnchdevt,pnchspace———►
►—,copydsn1,copydevt1,copyspace1—,copydsn2,copydevt2,copyspace2—,rcpydsn1,rcpydevt1,rcpyspace1—►
►—,rcpydsn2,rcpydevt2,rcpyspace2—,workdsn1,workdevt1,workspace1—,workdsn2,workdevt2,workspace2—►
►—,mapdsn,mapdevt,mapspace—,errdsn,errdevt,errspace—,filtrdsn,filtrdevt,filtrspace—)———►◄
```

DSNUTILS option descriptions

utility-id

Specifies a unique identifier for this utility within DB2.

This is an input parameter of type VARCHAR(16) in EBCDIC.

restart Specifies whether DB2 is to restart a current utility, and, if so, at what point the utility is to be restarted.

restart is an input parameter of type VARCHAR(8) in Unicode UTF-8, which must be translatable to allowable EBCDIC characters. Specify one of the following values for this parameter:

NO or null

Indicates that the utility job is new, not a restart. No other utility with the same utility identifier (UID) can exist.

The **default** is **null**.

CURRENT

Restarts the utility at the last commit point.

PHASE
Restarts the utility at the beginning of the currently stopped phase. Use the DISPLAY UTILITY to determine the currently stopped phase.

PREVIEW
Executes in PREVIEW mode the utility control statements that follow. While in PREVIEW mode, DB2 parses all utility control statements for syntax errors, but normal utility execution does not take place. If the syntax is valid, DB2 expands all LISTDEF lists and TEMPLATE data set name expressions that appear in SYSIN and prints the results to the SYSPRINT data set. DB2 evaluates and expands all LISTDEF definitions into an actual list of table spaces or index spaces. DB2 also evaluates TEMPLATE data set name expressions into actual data set names through variable substitution. DB2 also expands lists from the SYSLISTD DD and TEMPLATE data set name expressions from the SYSTEMPL DD that is referenced by a utility invocation.

Absence of the PREVIEW keyword turns off preview processing with one exception. The absence of this keyword does not override the PREVIEW JCL parameter which, if specified, remains in effect for the entire job step.

This option is identical to the PREVIEW JCL parameter.

utstmt Specifies the utility control statements.

This is an input parameter of type VARCHAR(32704) in EBCDIC.

retcode Specifies the utility highest return code.

This is an output parameter of type INTEGER.

utility-name
Specifies the utility that you want to invoke.

This is an input parameter of type VARCHAR(20) in EBCDIC.

Because DSNUTILS allows only a single utility here, dynamic support of data set allocation is limited. Specify only a single utility that requires data set allocation in the *utstmt* parameter. Select the utility name from the following list:

ANY[6]
CHECK DATA
CHECK INDEX
CHECK LOB
COPY
COPYTOCOPY
DIAGNOSE
LOAD
MERGECOPY
MODIFY RECOVERY
MODIFY STATISTICS
QUIESCE
REBUILD INDEX
RECOVER
REORG INDEX

6. Use ANY to indicate that TEMPLATE dynamic allocation is to be used. This value suppresses the dynamic allocation that is normally performed by DSNUTILS.

REORG LOB
REORG TABLESPACE
REPAIR
REPORT RECOVERY
REPORT TABLESPACESET
RUNSTATS INDEX
RUNSTATS TABLESPACE
STOSPACE
UNLOAD

Recommendation: Invoke DSNUTILS with a *utility-name* of ANY and omit all of the *xxx*dsn, *xxx*devt, and *xxx*space parameters. Use TEMPLATE statements to allocate the data sets.

recdsn Specifies the cataloged data set name that is required by LOAD for input, or by REORG TABLESPACE as the unload data set. *recdsn* is required for LOAD. It is also required for REORG TABLESPACE unless you also specified NOSYSREC or SHRLEVEL CHANGE. If you specify *recdsn*, the data set is allocated to the SYSREC DD name.

This is an input parameter of type VARCHAR(54) in EBCDIC.

If you specified the INDDN parameter for LOAD, the specified *ddname* value **must** be SYSREC.

If you specify the UNLDDN parameter for REORG TABLESPACE, the specified *ddname* value **must** be SYSREC.

recdevt Specifies a unit address, a generic device type, or a user-assigned group name for a device on which the *recdsn* data set resides.

This is an input parameter of type CHAR(8) in EBCDIC.

recspace
Specifies the number of cylinders to use as the primary space allocation for the *recdsn* data set. The secondary space allocation is 10% of the primary space allocation.

This is an input parameter of type SMALLINT.

discdsn Specifies the cataloged data set name that is used by LOAD as a discard data set to hold records not loaded, and by REORG TABLESPACE as a discard data set to hold records that are not reloaded. If you specify *discdsn*, the data set is allocated to the SYSDISC DD name.

This is an input parameter of type VARCHAR(54) in EBCDIC.

If you specify the DISCARDDN parameter for LOAD or REORG TABLESPACE, the specified *ddname* value **must** be SYSDISC.

discdevt
Specifies a unit address, a generic device type, or a user-assigned group name for a device on which the *discdsn* data set resides.

This is an input parameter of type CHAR(8) in EBCDIC.

discspace
Specifies the number of cylinders to use as the primary space allocation for the *discdsn* data set. The secondary space allocation is 10% of the primary space allocation.

This is an input parameter of type SMALLINT.

pnchdsn
Specifies the cataloged data set name that REORG TABLESPACE UNLOAD EXTERNAL or REORG TABLESPACE DISCARD uses to hold the generated LOAD utility control statements. If you specify a value for *pnchdsn*, the data set is allocated to the SYSPUNCH DD name.

This is an input parameter of type VARCHAR(54) in EBCDIC.

If you specify the PUNCHDDN parameter for REORG TABLESPACE, the specified *ddname* value **must** be SYSPUNCH.

pnchdevt
Specifies a unit address, a generic device type, or a user-assigned group name for a device on which the *pnchdsn* data set resides.

This is an input parameter of type CHAR(8) in EBCDIC.

pnchspace
Specifies the number of cylinders to use as the primary space allocation for the *pnchdsn* data set. The secondary space allocation is 10% of the primary space allocation.

This is an input parameter of type SMALLINT.

copydsn1
Specifies the name of the required target (output) data set, which is needed when you specify the COPY, COPYTOCOPY, or MERGECOPY utilities. It is optional for LOAD and REORG TABLESPACE. If you specify *copydsn1*, the data set is allocated to the SYSCOPY DD name.

This is an input parameter of type VARCHAR(54) in EBCDIC.

If you specify the COPYDDN parameter for COPY, COPYTOCOPY, MERGECOPY, LOAD, or REORG TABLESPACE, the specified *ddname1* value **must** be SYSCOPY.

copydevt1
Specifies a unit address, a generic device type, or a user-assigned group name for a device on which the *copydsn1* data set resides.

This is an input parameter of type CHAR(8) in EBCDIC.

copyspace1
Specifies the number of cylinders to use as the primary space allocation for the *copydsn1* data set. The secondary space allocation is 10% of the primary space allocation.

This is an input parameter of type SMALLINT.

copydsn2
Specifies the name of the cataloged data set that is used as a target (output) data set for the backup copy. It is optional for COPY, COPYTOCOPY, MERGECOPY, LOAD, and REORG TABLESPACE. If you specify *copydsn2*, the data set is allocated to the SYSCOPY2 DD name.

This is an input parameter of type VARCHAR(54) in EBCDIC.

If you specify the COPYDDN parameter for COPY, COPYTOCOPY, MERGECOPY, LOAD, or REORG TABLESPACE, the specified *ddname2* value **must** be SYSCOPY2.

copydevt2
Specifies a unit address, a generic device type, or a user-assigned group name for a device on which the *copydsn2* data set resides.

This is an input parameter of type CHAR(8) in EBCDIC.

copyspace2
Specifies the number of cylinders to use as the primary space allocation for the *copydsn2* data set. The secondary space allocation is 10% of the primary space allocation.

This is an input parameter of type SMALLINT.

rcpydsn1
Specifies the name of the cataloged data set that is required as a target (output) data set for the remote site primary copy. It is optional for COPY, COPYTOCOPY, LOAD, and REORG TABLESPACE. If you specify*rcpydsn1*, the data set is allocated to the SYSRCPY1 DD name.

This is an input parameter of type VARCHAR(54) in EBCDIC.

If you specified the RECOVERYDDN parameter for COPY, COPYTOCOPY, MERGECOPY, LOAD, or REORG TABLESPACE, the specified *ddname1* value **must** be SYSRCPY1.

rcpydevt1
Specifies a unit address, a generic device type, or a user-assigned group name for a device on which the *rcpydsn1* data set resides.

This is an input parameter of type CHAR(8) in EBCDIC.

rcpyspace1
Specifies the number of cylinders to use as the primary space allocation for the *rcpydsn1* data set. The secondary space allocation is 10% of the primary space allocation.

This is an input parameter of type SMALLINT.

rcpydsn2
Specifies the name of the cataloged data set that is required as a target (output) data set for the remote site backup copy. It is optional for COPY, COPYTOCOPY, LOAD, and REORG TABLESPACE. If you specify *rcpydsn2*, the data set is allocated to the **SYSRCPY2** DD name.

This is an input parameter of type VARCHAR(54) in EBCDIC.

If you specify the RECOVERYDDN parameter for COPY, COPYTOCOPY, MERGECOPY, LOAD, or REORG TABLESPACE, the specified *ddname2* value **must** be SYSRCPY2.

rcpydevt2
Specifies a unit address, a generic device type, or a user-assigned group name for a device on which the *rcpydsn2* data set resides.

This is an input parameter of type CHAR(8) in EBCDIC.

rcpyspace2
Specifies the number of cylinders to use as the primary space allocation for the *rcpydsn2* data set. The secondary space allocation is 10% of the primary space allocation

This is an input parameter of type SMALLINT.

workdsn1
Specifies the name of the cataloged data set that is required as a work data set for sort input and output. It is required for CHECK DATA, CHECK INDEX and REORG INDEX. It is also required for LOAD and REORG

TABLESPACE unless you also specify the SORTKEYS keyword. It is optional for REBUILD INDEX. If you specify *workdsn1*, the data set is allocated to the SYSUT1 DD name.

This is an input parameter of type VARCHAR(54) in EBCDIC.

If you specify the WORKDDN parameter for CHECK DATA, CHECK INDEX, LOAD, REORG INDEX, REORG TABLESPACE, or REBUILD INDEX, the specified *ddname* value **must** be SYSUT1.

workdevt1
Specifies a unit address, a generic device type, or a user-assigned group name for a device on which the *workdsn1* data set resides.

This is an input parameter of type CHAR(8) in EBCDIC.

workspace1
Specifies the number of cylinders to use as the primary space allocation for the *workdsn1* data set. The secondary space allocation is 10% of the primary space allocation.

This is an input parameter of type SMALLINT.

workdsn2
Specifies the name of the cataloged data set that is required as a work data set for sort input and output. It is required for CHECK DATA. It is also required if you use REORG INDEX to reorganize non-unique type 1 indexes. It is required for LOAD or REORG TABLESPACE unless you also specify the SORTKEYS keyword. If you specify *workdsn2*, the data set is allocated to the SORTOUT DD name.

This is an input parameter of type VARCHAR(54) in EBCDIC.

If you specify the WORKDDN parameter for CHECK DATA, LOAD, REORG INDEX, or REORG TABLESPACE, the specified *ddname* value **must** be SORTOUT.

workdevt2
Specifies a unit address, a generic device type, or a user-assigned group name for a device on which the *workdsn2* data set resides.

This is an input parameter of type CHAR(8) in EBCDIC.

workspace2
Specifies the number of cylinders to use as the primary space allocation for the *workdsn2* data set. The secondary space allocation is 10% of the primary space allocation.

This is an input parameter of type SMALLINT.

mapdsn
Specifies the name of the cataloged data set that is required as a work data set for error processing during LOAD with ENFORCE CONSTRAINTS. It is optional for LOAD. If you specify *mapdsn*, the data set is allocated to the SYSMAP DD name.

This is an input parameter of type VARCHAR(54) in EBCDIC.

If you specify the MAPDDN parameter for LOAD, the specified *ddname* value **must** be SYSMAP.

mapdevt
Specifies a unit address, a generic device type, or a user-assigned group name for a device on which the *mapdsn* data set resides.

This is an input parameter of type CHAR(8).

mapspace
Specifies the number of cylinders to use as the primary space allocation for the *mapdsn* data set. The secondary space allocation is 10% of the primary space allocation.

This is an input parameter of type SMALLINT.

errdsn Specifies the name of the cataloged data set that is required as a work data set for error processing. It is required for CHECK DATA, and it is optional for LOAD. If you specify *errdsn*, the data set is allocated to the SYSERR DD name.

This is an input parameter of type VARCHAR(54) in EBCDIC.

If you specify the ERRDDN parameter for CHECK DATA or LOAD, the specified *ddname* value **must** be SYSERR.

errdevt Specifies a unit address, a generic device type, or a user-assigned group name for a device on which the *errdsn* data set resides.

This is an input parameter of type CHAR(8) in EBCDIC.

errspace
Specifies the number of cylinders to use as the primary space allocation for the *errdsn* data set. The secondary space allocation is 10% of the primary space allocation.

This is an input parameter of type SMALLINT.

filtrdsn Specifies the name of the cataloged data set that is required as a work data set for error processing. It is optional for COPY CONCURRENT. If you specify *filtrdsn*, the data set is allocated to the FILTER DD name.

This is an input parameter of type VARCHAR(54) in EBCDIC.

If you specify the FILTERDDN parameter for COPY, the specified *ddname* value **must** be FILTER.

filtrdevt
Specifies a unit address, a generic device type, or a user-assigned group name for a device on which the *filtrdsn* data set resides.

This is an input parameter of type CHAR(8) in EBCDIC.

filtrspace
Specifies the number of cylinders to use as the primary space allocation for the *filtrdsn* data set. The secondary space allocation is 10% of the primary space allocation.

This is an input parameter of type SMALLINT.

Modifying the WLM-established address space for DSNUTILS

Add DSSPRINT, SYSIN, and SYSPRINT to the JCL procedure for starting the WLM-established address space in which DSNUTILS runs.

Requirement: You must allocate SYSIN and SYSPRINT in the procedure to temporarily store utility input statements and utility output messages. If you plan to invoke RUNSTATS and collect distribution statistics, you also need to allocate RNPRIN01.

Use JCL similar to the following sample PROC:

```
//*************************************************************
//*    JCL FOR RUNNING THE WLM-ESTABLISHED STORED PROCEDURES
//*    ADDRESS SPACE
//*       RGN     -- THE MVS REGION SIZE FOR THE ADDRESS SPACE.
//*       DB2SSN  -- THE DB2 SUBSYSTEM NAME.
//*       APPLENV -- THE MVS WLM APPLICATION ENVIRONMENT
//*                  SUPPORTED BY THIS JCL PROCEDURE.
//*
//*    IMPORTANT: You must use the value 1 in this EXEC card:
//*       IEFPROC EXEC PGM=DSNX9WLM,REGION=&RGN,TIME=NOLIMIT,
//*       PARM='&DB2SSN,1,&APPLENV'
//*
//*************************************************************
//DSNWLM   PROC RGN=0K,APPLENV=WLMENV1,DB2SSN=DSN
//IEFPROC EXEC PGM=DSNX9WLM,REGION=&RGN,TIME=NOLIMIT,
//        PARM='&DB2SSN,1,&APPLENV'
//STEPLIB  DD  DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
//         DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD
//UTPRINT  DD  SYSOUT=*
//RNPRIN01 DD SYSOUT=*
//DSSPRINT DD  SYSOUT=*
//SYSIN    DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSPRINT DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
```

Sample program for calling DSNUTILS

Three example programs calling DSNUTILS are shipped in SDSNSAMP.

- DSNTEJ6U: A DSNUTILS caller that uses PL/I. Job DSNTEJ6U compiles, link-edits, binds, and runs sample PL/I program DSN8EPU, which invokes the DSNUTILS stored procedure to execute an utility.
- DSNTEJ6V: A DSNUTILS caller that uses C++. Job DSNTEJ6V compiles, link-edits, binds, and runs sample C++ program DSN8EE1, which invokes the DSNUTILS stored procedure to execute an utility.
- DSNTEJ80: A DSNUTILS caller that uses C and ODBC. You can use this sample to compile, pre-link, link-edit, and execute the sample application DSN8OIVP, which you can use to verify that your DB2 ODBC installation is correct.

DSNUTILS output

DB2 creates the result set according to the DECLARE statement that is shown under "Example of declaring a cursor to select from SYSPRINT" on page 800.

Output from a successful execution of the DSNTEJ6U sample job or an equivalent job lists the specified parameters followed by the messages that are generated by the DB2 DIAGNOSE DISPLAY MEPL utility.

If DSNUTILB abends, the abend codes are returned as DSNUTILS return codes.

DSNUTILU stored procedure

The DSNUTILU stored procedure enables you to provide control statements in Unicode UTF-8 characters instead of EBCDIC characters to execute DB2 utilities from a DB2 application program. To use the stored procedure DSNUTILU, input data sets for the utility control statements can begin with the following Unicode characters:

- A Unicode UTF-8 blank (X'20')
- A Unicode UTF-8 dash (X'2D')
- Upper case Unicode UTF-8 ″A″ through ″Z″ (X'41' through X'5A')

When called, DSNUTILU performs the following actions:

- Translates the values that are specified for *utility-id* and *restart* to EBCDIC
- Creates the utility input (SYSIN) stream for control statements that use Unicode characters
- Invokes DB2 utilities (Program DSNUTILB)
- Deletes all the rows that are currently in the created temporary table (SYSIBM.SYSPRINT)
- Captures the utility output stream (SYSPRINT) into a created temporary table (SYSIBM.SYSPRINT)
- Declares a cursor to select from SYSPRINT:

```
DECLARE SYSPRINT CURSOR WITH RETURN FOR
  SELECT SEQNO, TEXT FROM SYSPRINT
  ORDER BY SEQNO;
```

- Opens the SYSPRINT cursor and returns

The calling program then performs fetches from the returned result set to obtain the captured utility output.

The BIND PACKAGE statement for the DSNUTILU stored procedure determines the character set of the resulting utility SYSPRINT output that is placed in the SYSIBM.SYSPRINT table. If ENCODING(EBCDIC) is specified, the SYSPRINT contents are in EBCDIC. If ENCODING(UNICODE) is specified, the SYSPRINT contents are in Unicode. The default install job, DSNTIJSG, is shipped with ENCODING(EBCDIC).

Environment for DSNUTILU

DSNUTILU **must** run in a WLM environment. TCB=1 is also required.

Authorization required for DSNUTILU

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNUTILU
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

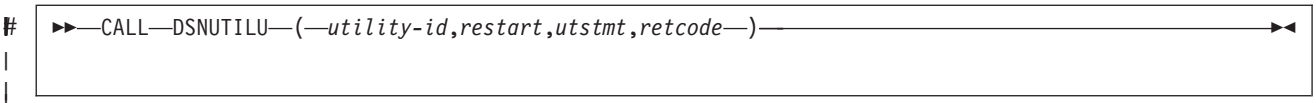
Then, to execute the utility, you must use a privilege set that includes the authorization to run the specified utility.

Control statement for DSNUTILU

DSNUTILU does not dynamically allocate data sets. The TEMPLATE utility control statement must be used to dynamically allocate data sets. Any utility that requires a sort must include the SORTDEVT keyword in the utility control statement. Use of the SORTNUM keyword is optional.

DSNUTILU stored procedure syntax diagram

The following syntax diagram shows the SQL CALL statement for invoking utilities as a stored procedure.



DSNUTILU option descriptions

utility-id

Specifies a unique identifier for this utility within DB2.

This is an input parameter of type VARCHAR(16) in Unicode UTF-8, which must be translatable to the following allowable EBCDIC characters:

- A - Z (upper and lower case)
- 0 - 9
- #, $, @, ¢, !, ^, or period (.)

restart Specifies whether DB2 is to restart a current utility, and, if so, at what point the utility is to be restarted.

restart is an input parameter of type VARCHAR(8) in Unicode UTF-8, which must be translatable to allowable EBCDIC characters. Specify one of the following values for this parameter:

NO or null

Indicates that the utility job is new, not a restart. No other utility with the same utility identifier (UID) can exist.

The **default** is **null**.

CURRENT

Restarts the utility at the last commit point.

PHASE

Restarts the utility at the beginning of the currently stopped phase. Use the DISPLAY UTILITY to determine the currently stopped phase.

PREVIEW

Executes in PREVIEW mode the utility control statements that follow. While in PREVIEW mode, DB2 parses all utility control statements for syntax errors, but normal utility execution does not take place. If the syntax is valid, DB2 expands all LISTDEF lists and TEMPLATE data set name expressions that appear in SYSIN and prints the results to the SYSPRINT data set. DB2 evaluates and expands all LISTDEFs into an actual list of table spaces or index spaces. DB2 also evaluates TEMPLATE data set name expressions into actual data set names through variable substitution. DB2 also expands lists from the SYSLISTD DD and TEMPLATE data set name expressions from the SYSTEMPL DD that is referenced by a utility invocation.

Absence of the PREVIEW keyword turns off preview processing with one exception. The absence of this keyword does not override the PREVIEW JCL parameter which, if specified, remains in effect for the entire job step.

This option is identical to the PREVIEW JCL parameter.

utstmt Specifies the utility control statements.

utstmt is an input parameter of type VARCHAR(32704) in UNICODE UTF-8.

retcode Specifies the utility highest return code.

retcode is an output parameter of type INTEGER.

Modifying the WLM-established address space for DSNUTILU

Add DSSPRINT, SYSIN, and SYSPRINT to the JCL procedure for starting the WLM-established address space, in which DSNUTILU runs. You must allocate SYSIN and SYSPRINT in the procedure to temporarily store utility input statements and utility output messages. If you plan to invoke RUNSTATS and collect distribution statistics, you also need to allocate RNPRIN01.

Use JCL similar to the sample PROC in Figure 150.

```
//*************************************************************
     //*    JCL FOR RUNNING THE WLM-ESTABLISHED STORED PROCEDURES
     //*    ADDRESS SPACE
     //*       RGN     -- THE MVS REGION SIZE FOR THE ADDRESS SPACE.
     //*       DB2SSN  -- THE DB2 SUBSYSTEM NAME.
     //*       APPLENV -- THE MVS WLM APPLICATION ENVIRONMENT
     //*                  SUPPORTED BY THIS JCL PROCEDURE.
     //*
     //*************************************************************
     //DSNWLM   PROC RGN=0K,APPLENV=WLMENV1,DB2SSN=DSN
     //IEFPROC EXEC PGM=DSNX9WLM,REGION=&RGN,TIME=NOLIMIT,
     //        PARM='&DB2SSN,1,&APPLENV'
     //STEPLIB  DD  DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
     //         DD  DISP=SHR,DSN=DSN!!0.SDSNLOAD
     //UTPRINT  DD  SYSOUT=*
     //RNPRIN01 DD  SYSOUT=*
     //DSSPRINT DD  SYSOUT=*
     //SYSIN    DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
     //SYSPRINT DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
```

Figure 150. Sample PROC for running the WLM-established stored procedures

Sample program for calling DSNUTILU

The following sample program calls DSNUTILU and is shipped in SDSNSAMP:

DSNTEJ6R - A DSNUTILU caller using PL/I. Job DSNTEJ6R compiles, link-edits, binds, and runs sample C-language caller program DSN8ED8, which invokes the DSNUTILU stored procedure to execute a utility.

DSNUTILU output

DB2 creates the result set according to the DECLARE statement shown on "Example of declaring a cursor to select from SYSPRINT" on page 800

Output from a successful execution of the DSNTEJ6R sample job or an equivalent job lists the specified parameters, followed by the messages that are generated by the DB2 DIAGNOSE DISPLAY MEPL utility.

The Control Center table space and index information stored procedure (DSNACCQC)

The information under this heading is Product-sensitive Programming Interface and Associated Guidance Information.

Restriction: The DSNACCQC stored procedure has been deprecated.

DSNACCQC is a sample stored procedure that gives you information about your table spaces and indexes. You can use DSNACCQC to obtain the following types of information:

- Table spaces and indexes on which RUNSTATS needs to be run
- Table spaces and indexes on which the STOSPACE utility has not been run
- Table spaces and indexes that exceed the primary space allocation
- Table spaces with more than a user-specified percentage of relocated rows
- Table spaces with more than a user-specified percentage of space that is occupied by dropped tables
- Table spaces with table space locking
- Simple table spaces with more than one table
- Indexes with clustering problems
- Indexes with more than a user-specified number of index levels
- Indexes with more than a user-specified LEAFDIST value
- Type 1 indexes
- Indexes with long RID chains that are not unique
- Indexes that are not used in static SQL statements

Environment for DSNACCQC

DSNACCQC runs in a WLM-established stored procedures address space.

Authorization required for DSNACCQC

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

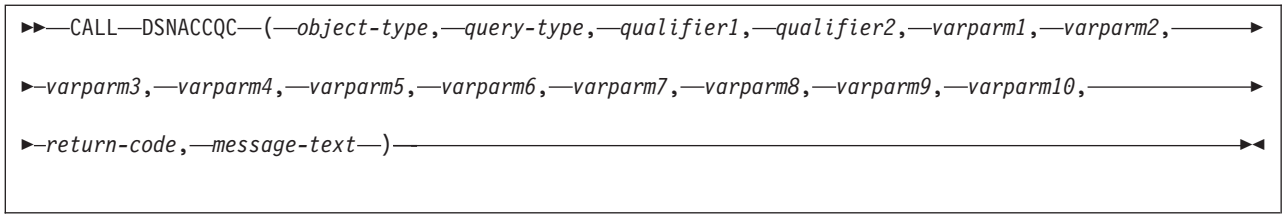
- The EXECUTE privilege on the package for DSNACCQC
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

The owner of the package or plan that contains the CALL statement must also have SELECT authority on the following catalog table spaces:

- SYSIBM.SYSINDEXES
- SYSIBM.SYSINDEXPART
- SYSIBM.SYSPACKDEP
- SYSIBM.SYSPLANDEP
- SYSIBM.SYSTABLEPART
- SYSIBM.SYSTABLES
- SYSIBM.SYSTABLESPACE

DSNACCQC syntax diagram

The following syntax diagram shows the SQL CALL statement for invoking DSNACCQC. Because the linkage convention for DSNACCQC is GENERAL, you cannot pass null values for input parameters. For character parameters that you are not using, specify an empty string ('').



DSNACCQC option descriptions

object-type
Specifies whether you want information about indexes or about table spaces. *object-type* is an input parameter of type INTEGER. The contents must be one of the following values:

0 Obtain information about indexes

1 Obtain information about table spaces

query-type
Specifies the type of information that you want to obtain. *query-type* is an input parameter of type INTEGER.

If *object-type* is 0 (indexes), *query-type* must be one of the following values:

0 Obtains information about indexes on which RUNSTATS needs to be run.

1 Obtains information about indexes with clustering problems.

2 Obtains information about indexes with more than a user-specified number of levels.

3 Obtains information about indexes with a LEAFDIST value that is higher than a user-specified percentage.

4 Obtains information about type 1 indexes.

5 Obtains information about indexes with long RID chains that are not unique.

6 Obtains information about indexes that are not used in static SQL statements.

7 Obtains information about indexes on which the STOSPACE utility has not been run.

8 Obtains information about indexes that have exceeded their allocated primary space quantity.

If *object-type* is 1 (table spaces), *query-type* must be one of the following values:

0 Obtains information about table spaces on which RUNSTATS needs to be run.

1 Obtains information about table spaces with more than a user-specified percentage of relocated rows.

2 Obtains information about table spaces with more than a user-specified percentage of space that is occupied by dropped tables.

3 Obtains information about table spaces with table space locking.

4 Obtains information about simple table spaces with more than one table.

5 Obtains information about table spaces on which the STOSPACE utility has not been run.

6 Obtains information about table spaces that have exceeded their allocated primary space quantity.

qualifier1
Narrows the search for objects that match *query-type* to a specified set of database names. *qualifier1* is an input parameter of type VARCHAR(255).

The format of this parameter is the same as the format of *pattern-expression* in an SQL LIKE predicate. *pattern-expression* is described in Chapter 4 of *DB2 SQL Reference*.

For example, to obtain information about table spaces or indexes in all databases with names that begin with ACCOUNT, specify this value for *qualifier1:*

```
ACCOUNT%
```

qualifier2
Narrows the search for objects that match *query-type* to a specified set of creator names. A creator name is the value of column CREATOR in SYSIBM.SYSTABLESPACE for table space queries, or SYSIBM.SYSINDEXES for index queries. *qualifier2* is an input parameter of type VARCHAR(255).

The format of this parameter is the same as the format of *pattern-expression* in an SQL LIKE predicate. *pattern-expression* is described in Chapter 4 of *DB2 SQL Reference*.

For example, to obtain information about table spaces or indexes with creators that begin with DSN8, specify this value for *qualifier2:*

```
DSN8%
```

varparm1, *varparm2*, *varparm3*
The meanings of these parameters vary with *object-type* and *query-type*. See Table 152 on page 816 for the meaning of each parameter for table space queries. See Table 153 on page 816 for the meaning of each parameter for index queries.

These are input parameters of type VARCHAR(255).

varparm4 **through** *varparm10*
These variables are reserved for future use. Specify an empty string ('') for each parameter value.

These are input parameters of type VARCHAR(255).

return-code
Specifies the return code from the DSNACCQC call, which is one of the following values:

0 DSNACCQC executed successfully.

12 An error occurred during DSNACCQC execution.

return-code is an output parameter of type INTEGER.

message-text
If an error occurs while DSNACCQC executes, contains information about the error. If the error is an SQL error, *message-text* also contains the formatted SQLCA. The message text consists of from one to eleven lines, each with a length of 121 bytes. The last byte of each line is a new-line character. *message-text* is an output parameter of type VARCHAR(1331).

Table 152. Variable input parameter values for DSNACCQC table space queries

query- type	**Parameter**	**Value**
0	*varparm1*	Specifies the table spaces for which to collect information. This is a timestamp in character format (*yyyy-mm-dd-hh.mm.ss.nnnnnn*). Information is collected for table spaces if RUNSTATS was run on them before this time or was never run.
0	*varparm2*	Not used. Specify an empty string ('').
0	*varparm3*	Not used. Specify an empty string ('').
1	*varparm1*	Character representation of a number between 0 and 100, which indicates the maximum acceptable percentage of relocated table rows. DSNACCQC returns information about table spaces for which (((FARINDREF+NEARINDREF)/CARD)*100)>*varparm1*.
1	*varparm2*	Not used. Specify an empty string ('').
1	*varparm3*	Not used. Specify an empty string ('').
2	*varparm1*	Character representation of a number between 0 and 100, which indicates the maximum acceptable percentage space that is occupied by rows of dropped tables. DSNACCQC returns information about table spaces for which PERCDROP>*varparm1*.
2	*varparm2*	Not used. Specify an empty string ('').
2	*varparm3*	Not used. Specify an empty string ('').
3	*varparm1*	Not used. Specify an empty string ('').
3	*varparm2*	Not used. Specify an empty string ('').
3	*varparm3*	Not used. Specify an empty string ('').
4	*varparm1*	Not used. Specify an empty string ('').
4	*varparm2*	Not used. Specify an empty string ('').
4	*varparm3*	Not used. Specify an empty string ('').
5	*varparm1*	Not used. Specify an empty string ('').
5	*varparm2*	Not used. Specify an empty string ('').
5	*varparm3*	Not used. Specify an empty string ('').
6	*varparm1*	Not used. Specify an empty string ('').
6	*varparm2*	Not used. Specify an empty string ('').
6	*varparm3*	Not used. Specify an empty string ('').

Table 153. Variable input parameter values for DSNACCQC index queries

query- type	**Parameter**	**Value**
0	*varparm1*	Specifies the indexes for which to collect information. This is a timestamp in character format (*yyyy-mm-dd-hh.mm.ss.nnnnnn*). Information is collected for indexes if RUNSTATS was run on them before this time or was never run.
0	*varparm2*	Not used. Specify an empty string ('').
0	*varparm3*	Not used. Specify an empty string ('').
1	*varparm1*	Character representation of a number between 0 and 100, which indicates the maximum acceptable percentage of table rows that are far from their optimal position. DSNACCQC returns information about indexes for which ((FAROFFPOSF/CARDF)*100)>*varparm1*.

Table 153. Variable input parameter values for DSNACCQC index queries (continued)

query- type	**Parameter**	**Value**
1	*varparm2*	Character representation of a number between 0 and 100, which indicates the maximum acceptable percentage of table rows that are near but not at their optimal position. DSNACCQC returns information about indexes for which ((NEAROFFPOSF/CARDF)*100)>*varparm2*.
1	*varparm3*	Character representation of a number between 0 and 100, which indicates the minimum acceptable percentage of table rows that are in clustering order. DSNACCQC returns information about indexes for which CLUSTERRATIO<*varparm3*.
2	*varparm1*	Character representation of a number that indicates the maximum acceptable number of index levels. DSNACCQC returns information about indexes for which NLEVELS>*varparm1*.
2	*varparm2*	Not used. Specify an empty string ('').
2	*varparm3*	Not used. Specify an empty string ('').
3	*varparm1*	Character representation of a number that indicates the maximum acceptable value for 100 times the average number of leaf pages between successive active leaf pages of the index. DSNACCQC returns information about indexes for which LEAFDIST>*varparm1*.
3	*varparm2*	Not used. Specify an empty string ('').
3	*varparm3*	Not used. Specify an empty string ('').
4	*varparm1*	Not used. Specify an empty string ('').
4	*varparm2*	Not used. Specify an empty string ('').
4	*varparm3*	Not used. Specify an empty string ('').
5	*varparm1*	Character representation of a number that indicates the maximum acceptable average length for RID chains. DSNACCQC returns information about indexes for which ((CARDF*1.0)/FULLKEYCARDF)>*varparm1*.
5	*varparm2*	Not used. Specify an empty string ('').
5	*varparm3*	Not used. Specify an empty string ('').
6	*varparm1*	Not used. Specify an empty string ('').
6	*varparm2*	Not used. Specify an empty string ('').
6	*varparm3*	Not used. Specify an empty string ('').
7	*varparm1*	Not used. Specify an empty string ('').
7	*varparm2*	Not used. Specify an empty string ('').
7	*varparm3*	Not used. Specify an empty string ('').
8	*varparm1*	Not used. Specify an empty string ('').
8	*varparm2*	Not used. Specify an empty string ('').
8	*varparm3*	Not used. Specify an empty string ('').

Example of DSNACCQC invocation

Suppose that you want information about indexes on which RUNSTATS has never been run. You want information about indexes in databases whose names begin with DSNCC only. The parameter declarations and DSNACCQC call look like

those in Figure 151:

```
DCL OBJTYPE    FIXED BIN(31);
DCL QUERY      FIXED BIN(31);
DCL DBQUAL     CHAR(255) VARYING;
DCL CREATQUAL  CHAR(255) VARYING;
DCL VARPARM1   CHAR(255) VARYING;
DCL VARPARM2   CHAR(255) VARYING;
DCL VARPARM3   CHAR(255) VARYING;
DCL VARPARM4   CHAR(255) VARYING;
DCL VARPARM5   CHAR(255) VARYING;
DCL VARPARM6   CHAR(255) VARYING;
DCL VARPARM7   CHAR(255) VARYING;
DCL VARPARM8   CHAR(255) VARYING;
DCL VARPARM9   CHAR(255) VARYING;
DCL VARPARM10  CHAR(255) VARYING;
DCL RC         FIXED BIN(31);
DCL MSGTEXT    CHAR(1331) VARYING;
DCL IXTABLE SQL TYPE IS RESULT_SET_LOCATOR VARYING;

OBJTYPE=0;
QUERY=0;
DBQUAL='DSNCC%';
CREATQUAL='%';
VARPARM1='0001-01-01-00.00.00.000000';
VARPARM2='';
VARPARM3='';
VARPARM4='';
VARPARM5='';
VARPARM6='';
VARPARM7='';
VARPARM8='';
VARPARM9='';
VARPARM10='';

EXEC SQL CALL SYSPROC.DSNACCQC(:OBJTYPE, :QUERY, :DBQUAL,
 :CREATQUAL, :VARPARM1, :VARPARM2,  :VARPARM3,
 :VARPARM4,  :VARPARM5, :VARPARM6,  :VARPARM7,
 :VARPARM8,  :VARPARM9, :VARPARM10,
 :RC, :MSGTEXT);
```

Figure 151. Example of DSNACCQC invocation

DSNACCQC output

In addition to the output parameters described in "DSNACCQC option descriptions" on page 814, DSNACCQC returns one result set. The format of the result set varies, depending on whether you are retrieving index information (*object-type*=0) or table space information (*object-type*=1).

Table 154 shows the columns of a result set row and the DB2 catalog table that is the source of information for each column for table space queries.

Table 154. Result set columns for DSNACCQC table space queries

Column name	Data type	DB2 catalog table that is the data source
NAME	CHAR(8)	SYSTABLESPACE
CREATOR	VARCHAR(128)	SYSTABLESPACE
BPOOL	CHAR(8)	SYSTABLESPACE
LOCKRULE	CHAR(1)	SYSTABLESPACE
LOCKMAX	INTEGER	SYSTABLESPACE

Table 154. Result set columns for DSNACCQC table space queries (continued)

Column name	Data type	DB2 catalog table that is the data source
CLOSERULE	CHAR(1)	SYSTABLESPACE
ENCODING_SCHEME	CHAR(1)	SYSTABLESPACE
LOCKPART	CHAR(1)	SYSTABLESPACE
MAXROWS	SMALLINT	SYSTABLESPACE
PARTITIONS	SMALLINT	SYSTABLESPACE
TYPE	CHAR(1)	SYSTABLESPACE
SEGSIZE	SMALLINT	SYSTABLESPACE
SPACE	INTEGER	SYSTABLESPACE
NTABLES	SMALLINT	SYSTABLESPACE
STATUS	CHAR(1)	SYSTABLESPACE
STATSTIME	TIMESTAMP	SYSTABLESPACE
ERASERULE	CHAR(1)	SYSTABLESPACE
DBNAME	CHAR(8)	SYSTABLESPACE
DSETPASS	CHAR(8)	SYSTABLESPACE
LOG	CHAR(1)	SYSTABLESPACE
DSSIZE	INTEGER	SYSTABLESPACE
SBCS_CCSID	INTEGER	SYSTABLESPACE

Table 155 shows the columns of a result set row and the DB2 catalog table that is the source of information for index queries.

Table 155. Result set columns for DSNACCQC index queries

Column name	Data type	DB2 catalog table that is the data source
CREATOR	VARCHAR(128)	SYSINDEXES
NAME	VARCHAR(128)	SYSINDEXES
TBCREATOR	VARCHAR(128)	SYSINDEXES
TBNAME	VARCHAR(128)	SYSINDEXES
UNIQUERULE	CHAR(1)	SYSINDEXES
INDEXTYPE	CHAR(1)	SYSINDEXES
INDEXSPACE	CHAR(8)	SYSINDEXES
CLUSTERING	CHAR(1)	SYSINDEXES
ERASERULE	CHAR(1)	SYSINDEXES
CLOSERULE	CHAR(1)	SYSINDEXES
COLCOUNT	SMALLINT	SYSINDEXES
DBID	SMALLINT	SYSINDEXES
DBNAME	CHAR(8)	SYSINDEXES
BPOOL	CHAR(8)	SYSINDEXES
PGSIZE	SMALLINT	SYSINDEXES
DSETPASS	CHAR(8)	SYSINDEXES

Table 155. Result set columns for DSNACCQC index queries (continued)

Column name	Data type	DB2 catalog table that is the data source
PIECESIZE	INTEGER	SYSINDEXES
COPY	CHAR(1)	SYSINDEXES
PARTITION_COUNT	INTEGER	SYSINDEXPART[1]

Notes:

1. The value of PARTITION_COUNT is COUNT(DISTINCT PARTITION) for partitioning indexes or 0 for nonpartitioning indexes. The PARTITION column is in SYSIBM.SYSINDEXPART.

To obtain the information from the result set, you can write your client program to retrieve information from one result set with known contents. However, for greater flexibility, you might want to write your client program to retrieve data from an unknown number of result sets with unknown contents. Both techniques are shown in Part 6 of *DB2 Application Programming and SQL Guide*.

The Control Center partition information stored procedure (DSNACCAV)

The information under this heading is Product-sensitive Programming Interface and Associated Guidance Information.

Restriction: The DSNACCAV stored procedure has been deprecated.

DSNACCAV is a sample stored procedure that gives you information about partitions in your table spaces and indexes. You can use DSNACCAV to obtain the following types of information:

- Partitions that need to be image copied
- Partitions that are in a restricted state
- Partitions on which RUNSTATS needs to be run
- Partitions on which REORG needs to be run
- Partitions that exceed a user-specified number of extents

Environment for DSNACCAV

DSNACCAV runs in a WLM-established stored procedures address space.

Authorization required for DSNACCAV

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNACCAV
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

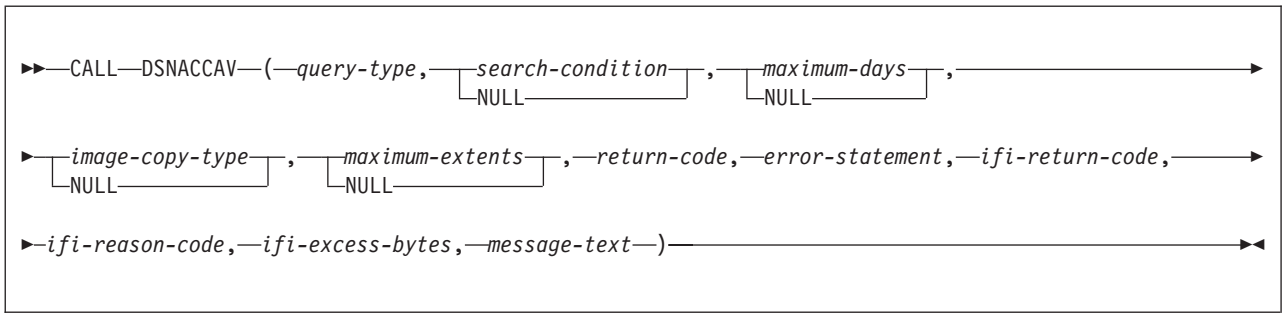
The owner of the package or plan that contains the CALL statement must also have SELECT authority on the following catalog table spaces:

- SYSIBM.SYSCOPY
- SYSIBM.SYSINDEXES
- SYSIBM.SYSINDEXPART
- SYSIBM.SYSTABLEPART

- SYSIBM.SYSTABLES
- SYSIBM.SYSTABLESPACE

DSNACCAV syntax diagram

The following syntax diagram shows the SQL CALL statement for invoking DSNACCAV. Because the linkage convention for DSNACCAV is GENERAL WITH NULLS, if you pass parameters in host variables, you need to include a null indicator with every host variable. Null indicators for input host variables must be initialized before you execute the CALL statement.



DSNACCAV option descriptions

query-type
Specifies the type of information that you want to obtain. *query-type* is an input parameter of type VARCHAR(20). The contents must be one of the following values:

COPY TABLESPACE
Obtains information about table space partitions for which image copies need to be made.

COPY INDEX
Obtains information about index partitions for which image copies need to be made.

RESTRICT TABLESPACE
Obtains information about table space partitions that are in a restricted status.

RESTRICT INDEX
Obtains information about index partitions that are in a restricted status.

RUNSTATS TABLESPACE
Obtains information about table space partitions on which RUNSTATS needs to be run.

RUNSTATS INDEX
Obtains information about index partitions on which RUNSTATS needs to be run.

REORG TABLESPACE
Obtains information about table space partitions on which REORG needs to be run.

REORG INDEX
Obtains information about index partitions on which REORG needs to be run.

EXTENTS TABLESPACE
Obtains information about table space partitions that have used more than a user-specified number of extents.

EXTENTS INDEX
Obtains information about index partitions that have used more than a user-specified number of extents.

search-condition
Narrows the search for objects that match *query-type. search-condition* is an input parameter of type VARCHAR(4096).

The format of this parameter is the same as the format of *search-condition* in an SQL *where-clause.search-condition* is described in Chapter 4 of *DB2 SQL Reference*.

If the call is executed to obtain table space information, *search-condition* can include any column in SYSIBM.SYSTABLESPACE. If the call is executed to obtain index information, *search-condition* can include any column in SYSIBM.SYSINDEXES. Each column name must be preceded by the string 'A.'.

For example, to obtain information about table spaces with creator ADMF001, specify this value for *search-condition:*

```
A.CREATOR='ADMF001'
```

maximum-days
Specifies the maximum number of days that are to elapse between executions of the REORG, RUNSTATS, or COPY utility. DSNACCAV uses this value as the criterion for determining which table space or index partitions need to have the utility that you specified in *query-type* run against them. This value can be specified if *query-type* has one of the following values:
- COPY TABLESPACE
- COPY INDEX
- RUNSTATS TABLESPACE
- RUNSTATS INDEX
- REORG TABLESPACE
- REORG INDEX

maximum-days is an input parameter of type INTEGER.

image-copy-type
Specifies the types of image copies about which DSNACCAV is to give you information. This value can be specified if *query-type* is COPY TABLESPACE or COPY INDEX.*image-copy-type* is an input parameter of type CHAR(1). The contents must be one of the following values:

B Specifies that you want information about partitions for which the most recent image copy was either a full image copy or an incremental image copy

F Specifies that you want information about partitions for which the most recent image copy was a full image copy

I Specifies that you want information about partitions for which the most recent image copy was an incremental image copy

maximum-extents
Specifies the maximum number of extents that a table space or index partition is to use. This value can be specified if *query-type* is one of the following values:
- REORG TABLESPACE
- REORG INDEX
- EXTENTS TABLESPACE

- EXTENTS INDEX

maximum-extents is an input parameter of type INTEGER.

return-code
Specifies the return code from the DSNACCAV call, which is one of the following values:

0 DSNCCAV executed successfully.

12 An error occurred during DSNCCAV execution.

return-code is an output parameter of type INTEGER.

error-statement
If *return-code* is not 0, specifies the SQL statement or DB2 command that DB2 was executing when the error occurred. *error-statement* is an output parameter of type VARCHAR(8012).

ifi-return-code
When *query-type* is RESTRICT TABLESPACE, RESTRICT INDEX, COPY TABLESPACE, or REORG TABLESPACE, specifies the return code from the IFI call that submitted a DISPLAY DATABASE command to obtain information about restricted objects. *ifi-return-code* is an output parameter of type INTEGER.

ifi-reason-code
When *query-type* is RESTRICT TABLESPACE, RESTRICT INDEX, COPY TABLESPACE, or REORG TABLESPACE, specifies the reason code from the IFI call that submitted a DISPLAY DATABASE command to obtain information about restricted objects. *ifi-reason-code* is an output parameter of type INTEGER.

ifi-excess-bytes
When *query-type* is RESTRICT TABLESPACE, RESTRICT INDEX, COPY TABLESPACE, or REORG TABLESPACE, specifies the number of bytes that did not fit in the return area for the IFI call that submitted a DISPLAY DATABASE command to obtain information about restricted objects. *ifi-excess-bytes* is an output parameter of type INTEGER.

message-text
If an SQL error occurs while DSNACCAV executes, contains information about the error, including the formatted SQLCA. The message text consists of from one to ten lines, each with a length of 121 bytes. The last byte of each line is a new-line character. *message-text* is an output parameter of type VARCHAR(1210).

Example of DSNACCAV invocation

Suppose that you want information about table space partitions that are in a restricted status. You want information about table spaces that are in databases whose names begin with DSNCC only. The parameter declarations and DSNACCAV call looks like those in Figure 152 on page 824:

```
DCL QUERY      CHAR(20) VARYING;
DCL CRITERIA   CHAR(4096) VARYING;
DCL NUMDAYS    FIXED BIN(31);
DCL OPTYPE     CHAR(1) VARYING;
DCL EXTENTS    FIXED BIN(31);
DCL RC         FIXED BIN(31);
DCL STMT       CHAR(8012) VARYING;
DCL IFIRC      FIXED BIN(31);
DCL IFIREASON  FIXED BIN(31);
DCL IFIEXCESS  FIXED BIN(31);
DCL STMT       CHAR(8012) VARYING;
DCL MSGTEXT    CHAR(1331) VARYING;
DCL IND1 FIXED BIN(15);
DCL IND2 FIXED BIN(15);
DCL IND3 FIXED BIN(15);
DCL IND4 FIXED BIN(15);
DCL IND5 FIXED BIN(15);
DCL IND6 FIXED BIN(15);
DCL IND7 FIXED BIN(15);
DCL IND8 FIXED BIN(15);
DCL IND9 FIXED BIN(15);
DCL IND10 FIXED BIN(15);
DCL IND11 FIXED BIN(15);
DCL CMDMSG  SQL TYPE IS RESULT_SET_LOCATOR VARYING;
DCL TSTABLE SQL TYPE IS RESULT_SET_LOCATOR VARYING;

QUERY='RESTRICT TABLESPACE';
IND1=0;
Criteria='A.DBNAME LIKE ''DSNCC%''';
IND2=0;
numdays=0;
IND3=0;
optype='';
IND4=0;
extents=0;
IND5=0;

EXEC SQL CALL SYSPROC.DSNACCAV(:QUERY :IND1, :CRITERIA :IND2,
 :NUMDAYS :IND3, :OPTYPE :IND4, :EXTENTS :IND5,
 :RC :IND6, :STMT :IND7, :IFIRC :IND8,
 :IFIREASON :IND9, :IFIEXCESS :IND10, :MSGTEXT :IND11);
```

Figure 152. Example of DSNACCAV invocation

DSNACCAV output

In addition to the output parameters that are described in "DSNACCAV option descriptions" on page 821, DSNACCAV returns two result sets.

The first result set contains the text from commands that DB2 executes, formatted into 80-byte records.

Table 156 shows the format of the first result set.

Table 156. Result set row for DSNACCAV command output

Column name	Data type	Contents
RS_SEQUENCE	INTEGER	Sequence number of the output line
RS_DATA	CHAR(80)	A line of command output

The second result set contains partition information. The format of the second result set varies, depending on whether you request table space or index information.

Table 157 shows the columns of a result set row and the DB2 catalog table that is the source of information for each column for table space queries.

Table 157. Result set row for DSNACCAV table space queries

Column name	Data type	DB2 catalog table that is the data source
NAME	CHAR(8)	SYSTABLESPACE
CREATOR	CHAR(8)	SYSTABLESPACE
BPOOL	CHAR(8)	SYSTABLESPACE
LOCKRULE	CHAR(1)	SYSTABLESPACE
LOCKMAX	INTEGER	SYSTABLESPACE
CLOSERULE	CHAR(1)	SYSTABLESPACE
ENCODING_SCHEME	CHAR(1)	SYSTABLESPACE
LOCKPART	CHAR(1)	SYSTABLESPACE
MAXROWS	SMALLINT	SYSTABLESPACE
PARTITIONS	SMALLINT	SYSTABLESPACE
TYPE	CHAR(1)	SYSTABLESPACE
SEGSIZE	SMALLINT	SYSTABLESPACE
SPACE	INTEGER	SYSTABLESPACE
NTABLES	SMALLINT	SYSTABLESPACE
STATUS	CHAR(1)	SYSTABLESPACE
STATSTIME	TIMESTAMP	SYSTABLESPACE
ERASERULE	CHAR(1)	SYSTABLESPACE
DBNAME	CHAR(8)	SYSTABLESPACE
DSETPASS	CHAR(8)	SYSTABLESPACE
LOG	CHAR(1)	SYSTABLESPACE
DSSIZE	INTEGER	SYSTABLESPACE
PARTITION	SMALLINT	SYSTABLEPART
OPERATIONTIME	TIMESTAMP	SYSCOPY or SYSTABLEPART[1]
DAYS	INTEGER	SYSCOPY or SYSTABLEPART[2]
PERCOFFPOS	SMALLINT	SYSINDEXPART[3]
PERCINDREF	SMALLINT	SYSTABLEPART[4]
PERCDROP	SMALLINT	SYSTABLEPART
EXTENTS	INTEGER	None[5]
REASON	CHAR(18)	None[6]
SBCS_CCSID	INTEGER	SYSTABLESPACE

Table 157. Result set row for DSNACCAV table space queries (continued)

Column name	Data type	DB2 catalog table that is the data source

Notes:

1. If *query-type* is COPY TABLESPACE or REORG TABLESPACE, the value of OPERATIONTIME is the value of the TIMESTAMP column in SYSIBM.SYSCOPY.

   If *query-type* is RUNSTATS TABLESPACE, the value of OPERATIONTIME is the value of the TIMESTAMP column in SYSIBM.SYSTABLEPART.
2. DAYS is the number of days since the last invocation of the utility. This column is derived from the OPERATIONTIME column.
3. PERCOFFPOS=(NEAROFFPOSF+FAROFFPOSF)*100/CARDF
4. PERCINDREF=(NEARINDREF+FARINDREF)*100/CARD
5. EXTENTS is the number of data set extents that the partition is using.
6. REASON is the reason that the row is in the result set. See Table 158 for values of REASON for each value of *query-type*.

Table 158 shows the values of the REASON column for each *query-type* value for table space queries.

Table 158. Values of the REASON result set column for table space queries

query-type	REASON value	REASON meaning
COPY TABLESPACE	Status from DISPLAY DATABASE command output	Table space is in restricted status COPY
COPY TABLESPACE	DAYS	The number of days since the last COPY occurred exceeds the *maximum-days* value
RESTRICT TABLESPACE	Status from DISPLAY DATABASE command output	Table space is in restricted status
RUNSTATS TABLESPACE	LOAD	LOAD was run after RUNSTATS
RUNSTATS TABLESPACE	REORG	REORG was run after RUNSTATS
RUNSTATS TABLESPACE	RECOVER	RECOVER was run after RUNSTATS
RUNSTATS TABLESPACE	DAYS	The number of days since the last RUNSTATS occurred exceeds the *maximum-days* value

REORG TABLESPACE	LIMIT	One of the following reasons:
		• A clustering index meets this condition:
		`((NEAROFFPOSF+FAROFFPOSF)*100/CARDF)>10`
		• A partition meets either of these conditions:
		`((NEARINDREF+FARINDREF)* 100/CARD)>10`
		`PERCDROP>10`

REORG TABLESPACE	DAYS	The number of days since the last REORG occurred exceeds the *maximum-days* value

Table 158. Values of the REASON result set column for table space queries (continued)

query-type	**REASON value**	**REASON meaning**
REORG TABLESPACE	EXTENTS	Number of partition extents exceeds *maximum-extents* value
REORG TABLESPACE	Status from DISPLAY DATABASE command output	The table space is in restricted status REORP
EXTENTS TABLESPACE	EXTENTS	Number of partition extents exceeds *maximum-extents* value

Table 159 shows the columns of a result set row and the DB2 catalog table that is the source of information for each column for index queries.

Table 159. Result set row for DSNACCAV index queries

Column name	**Data type**	**DB2 catalog table that is the data source**
CREATOR	CHAR(8)	SYSINDEXES
NAME	VARCHAR(18)	SYSINDEXES
TBCREATOR	CHAR(8)	SYSINDEXES
TBNAME	VARCHAR(18)	SYSINDEXES
UNIQUERULE	CHAR(1)	SYSINDEXES
INDEXTYPE	CHAR(1)	SYSINDEXES
INDEXSPACE	CHAR(8)	SYSINDEXES
CLUSTERING	CHAR(1)	SYSINDEXES
ERASERULE	CHAR(1)	SYSINDEXES
CLOSERULE	CHAR(1)	SYSINDEXES
COLCOUNT	SMALLINT	SYSINDEXES
DBID	SMALLINT	SYSINDEXES
DBNAME	CHAR(8)	SYSINDEXES
BPOOL	CHAR(8)	SYSINDEXES
PGSIZE	SMALLINT	SYSINDEXES
DSETPASS	CHAR(8)	SYSINDEXES
PIECESIZE	INTEGER	SYSINDEXES
COPY	CHAR(1)	SYSINDEXES
PARTITIONS	SMALLINT	SYSINDEXPART[1]
PARTITION	SMALLINT	SYSINDEXPART
OPERATIONTIME	TIMESTAMP	SYSCOPY or SYSINDEXPART[2]
DAYS	INTEGER	SYSCOPY or SYSTABLEPART[3]
LEAFDIST	INTEGER	SYSINDEXPART
EXTENTS	INTEGER	None[4]
REASON	CHAR(18)	None[5]

Table 159. Result set row for DSNACCAV index queries (continued)

Column name	Data type	DB2 catalog table that is the data source

Notes:

1. PARTITIONS is derived from the PARTITION column through this SELECT statement:

```
SELECT IXNAME,IXCREATOR,MAX(PARTITION) AS PARTITIONS
  FROM SYSIBM.SYSINDEXPART
  GROUP BY IXNAME,IXCREATOR;
```

2. If *query-type* is COPY INDEX or REORG INDEX, the value of OPERATIONTIME is the value of the TIMESTAMP column in SYSIBM.SYSCOPY.

   If *query-type* is RUNSTATS INDEX, the value of OPERATIONTIME is the value of the TIMESTAMP column in SYSIBM.SYSINDEXPART.
3. DAYS is the number of days since the last invocation of the utility. This column is derived from the OPERATIONTIME column.
4. EXTENTS is the number of data set extents that the partition is using.
5. REASON is the reason that the row is in the result set. See Table 160 for values of REASON for each value of *query-type*.

Table 160 shows the values of the REASON column for each *query-type* value for index queries.

Table 160. Values of the REASON result set column for index queries

query-type	**REASON value**	**REASON meaning**
COPY INDEX	LIMIT	Index is in restricted status ICOPY
COPY INDEX	DAYS	The number of days since the last COPY occurred exceeds the *maximum-days* value
RESTRICT INDEX	Status from DISPLAY DATABASE command output	Index is in restricted status
RUNSTATS INDEX	TABLESPACE LOAD	LOAD was run on the associated table space after RUNSTATS
RUNSTATS INDEX	TABLESPACE REORG	REORG was run on the associated table space after RUNSTATS
RUNSTATS INDEX	TABLESPACE RECOVER	RECOVER was run on the associated table space after RUNSTATS
RUNSTATS INDEX	REBUILT	REBUILD was run after RUNSTATS
RUNSTATS INDEX	DAYS	The number of days since the last RUNSTATS occurred exceeds *maximum-days* value
REORG INDEX	LIMIT	LEAFDIST exceeds the recommended limit of 200
REORG INDEX	DAYS	The number of days since the last REORG occurred exceeds *maximum-days* value
REORG INDEX	EXTENTS	Number of partition extents exceeds *maximum-extents* value
EXTENTS INDEX	EXTENTS	Number of partition extents exceeds *maximum-extents* value

The number of rows that are returned in the second result set varies with *query-type*. Table 161 on page 829 shows the number and types of rows that are

returned from an invocation of DSNACCAV for each *query-type*.

Table 161. Rows of the second DSNACCAV result set for each query type

query-type	**Rows returned**
COPY TABLESPACE	One row for: • Each table space partition that has not been copied within the number of days that are specified by the *maximum-days* parameter • The most recent copy of each data set in a nonpartitioned table space • Each table space partition that is in COPY-pending status
COPY INDEX	One row for: • Each index space partition that has not been copied within the number of days that are specified by the *maximum-days* parameter • The most recent copy of each data set in a nonpartitioned index space • Each index space partition that is in ICOPY-pending status
RESTRICT TABLESPACE	One row for each table space in the subsystem that meets the criteria that are specified by the *search-criteria* parameter and is in a restricted status
RESTRICT INDEX	One row for each index in the subsystem that meets the criteria specified by the *search-criteria* parameter and is in a restricted status
RUNSTATS TABLESPACE	One row for: • Each table space partition on which LOAD, REORG, or RECOVER was run after the last time RUNSTATS was run • Each table space partition on which RUNSTATS was not run within the number of days that are specified by the *maximum-days* parameter
RUNSTATS INDEX	One row for: • Each index partition that is defined on a table on which LOAD, REORG, or RECOVER was run after the last time RUNSTATS was run • Each index partition on which RUNSTATS was not run within the number of days that are specified by the *maximum-days* parameter • Each index partition on which REBUILD was run after the last time RUNSTATS was run
REORG TABLESPACE	One row for: • Each table space partition that is in REORG-pending status • Each table space partition for which the number of data set extents exceeds the value that is specified by *maximum-extents* • Each table space partition for which the clustering index that is associated with the table has (NEAROFFPOSF+FAROFFPOSF)*100/CARDF>10 • Each table space partition for which (NEARINDREF+FARINDREF)*100/CARD>10 • Each table space partition for which PERCDROP>10 • Each table space partition on which REORG was not run within the number of days that are specified by the *maximum-days* parameter

Table 161. Rows of the second DSNACCAV result set for each query type (continued)

query-type	**Rows returned**
REORG INDEX	One row for: • Each index partition for which LEAFDIST>200 • Each index partition for which the number of data set extents exceeds the value that is specified by *maximum-extents* • Each index partition on which REORG was not run within the number of days specified by the *maximum-days* parameter
EXTENTS TABLESPACE	One row for each table space partition for which the number of data set extents exceeds the value that is specified by *maximum-extents*
EXTENTS INDEX	One row for each index partition for which the number of data set extents exceeds the value that is specified by *maximum-extents*

To obtain the information from the result sets, you can write your client program to retrieve information from two result sets with known contents. However, for greater flexibility, you might want to write your client program to retrieve data from an unknown number of result sets with unknown contents. Both techniques are shown in Part 6 of *DB2 Application Programming and SQL Guide*.

The DB2 real-time statistics stored procedure

The information under this heading is Product-sensitive Programming Interface and Associated Guidance Information.

The DSNACCOR stored procedure is a sample stored procedure that makes recommendations to help you maintain your DB2 databases. In particular, DSNACCOR performs these actions:

- Recommends when you should reorganize, image copy, or update statistics for table spaces or index spaces
- Indicates table spaces or index spaces that have exceeded their data set
- Indicates whether objects are in a restricted state

DSNACCOR uses data from the SYSIBM.TABLESPACESTATS and SYSIBM.INDEXSPACESTATS real-time statistics tables to make its recommendations. DSNACCOR provides its recommendations in a result set.

DSNACCOR uses the set of criteria that are shown in "DSNACCOR formulas for recommending actions" on page 840 to evaluate table spaces and index spaces. By default, DSNACCOR evaluates **all** table spaces and index spaces in the subsystem that have entries in the real-time statistics tables. However, you can override this default through input parameters.

Important information about DSNACCOR recommendations:

- DSNACCOR makes recommendations based on general formulas that require input from the user about the maintenance policies for a subsystem. These recommendations might not be accurate for every installation.
- If the real-time statistics tables contain information for only a small percentage of your DB2 subsystem, the recommendations that DSNACCOR makes might not be accurate for the entire subsystem.
- Before you perform any action that DSNACCOR recommends, ensure that the object for which DSNACCOR makes the recommendation is available, and that

the recommended action can be performed on that object. For example, before you can perform an image copy on an index, the index must have the COPY YES attribute.

Environment for DSNACCOR

DSNACCOR must run in a WLM-established stored procedure address space.

DSNACCOR creates and uses declared temporary tables. Therefore, before you can invoke DSNACCOR, you need to create a TEMP database and segmented table spaces in the TEMP database. For information about creating TEMP databases and table spaces, see CREATE DATABASE and CREATE TABLESPACE Chapter 5 of *DB2 SQL Reference*.

Before you can invoke DSNACCOR, the real-time statistics tables, SYSIBM.TABLESPACESTATS and SYSIBM.INDEXSPACESTATS, must exist, and the real-time statistics database must be started. See Appendix E, "Real-time statistics tables," on page 873 for information about the real-time statistics tables.

You should bind the package for DSNACCOR with isolation UR to avoid lock contention. You can find the installation steps for DSNACCOR in job DSNTIJSG.

Authorization required for DSNACCOR

To execute the CALL DSNACCOR statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

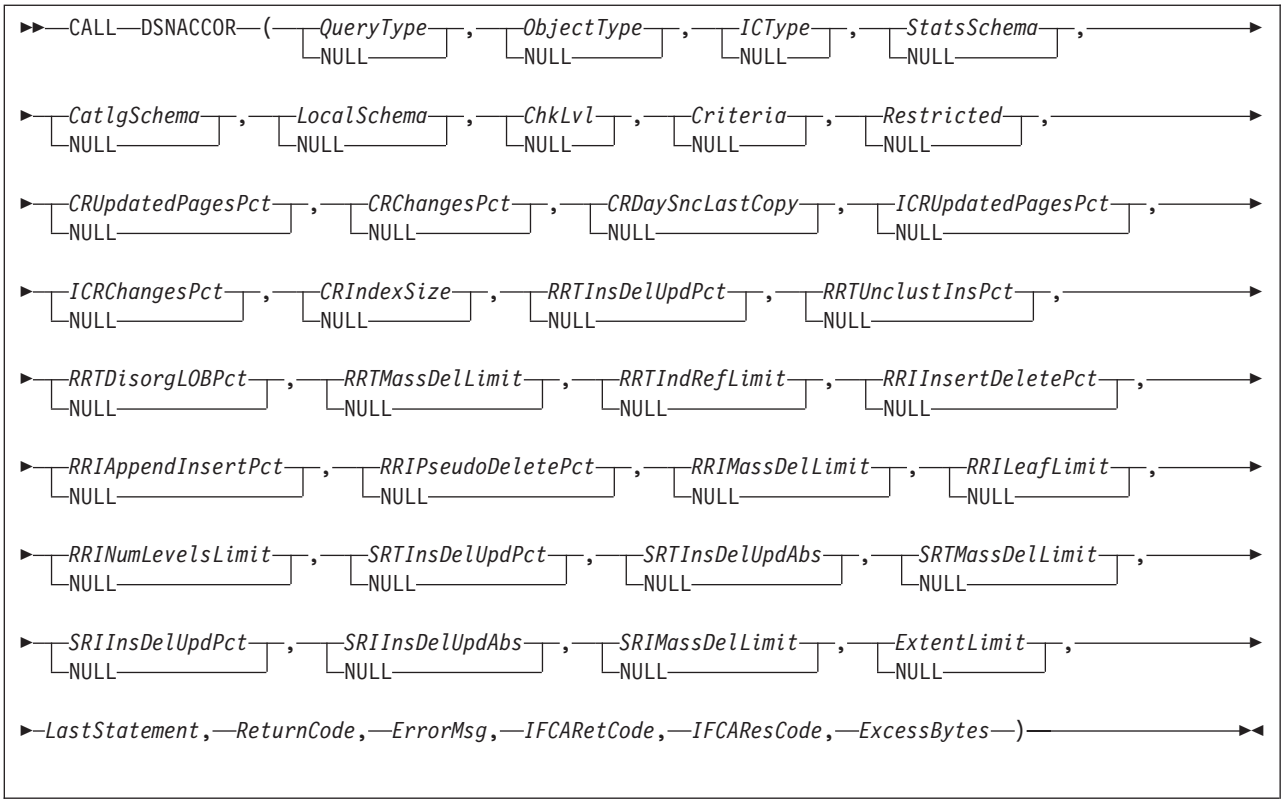
- The EXECUTE privilege on the package for DSNACCOR
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

The owner of the package or plan that contains the CALL statement must also have:

- SELECT authority on the real-time statistics tables
- The DISPLAY system privilege

DSNACCOR syntax diagram

The following syntax diagram shows the CALL statement for invoking DSNACCOR. Because the linkage convention for DSNACCOR is GENERAL WITH NULLS, if you pass parameters in host variables, you need to include a null indicator with every host variable. Null indicators for input host variables must be initialized before you execute the CALL statement.



DSNACCOR option descriptions

In the following option descriptions, the default value for an input parameter is the value that DSNACCOR uses if you specify a null value.

QueryType
Specifies the types of actions that DSNACCOR recommends. This field contains one or more of the following values. Each value is enclosed in single quotation marks and separated from other values by a space.

ALL Makes recommendations for all of the following actions.

COPY Makes a recommendation on whether to perform an image copy.

RUNSTATS Makes a recommendation on whether to perform RUNSTATS.

REORG Makes a recommendation on whether to perform REORG. Choosing this value causes DSNACCOR to process the EXTENTS value also.

EXTENTS Indicates when data sets have exceeded a user-specified extents limit.

RESTRICT Indicates which objects are in a restricted state.

QueryType is an input parameter of type VARCHAR(40). The default is **ALL**.

ObjectType
Specifies the types of objects for which DSNACCOR recommends actions:

ALL Table spaces and index spaces.

TS Table spaces only.

IX Index spaces only.

ObjectType is an input parameter of type VARCHAR(3). The default is **ALL**.

ICType
Specifies the types of image copies for which DSNACCOR is to make recommendations:

F Full image copy.

I Incremental image copy. This value is valid for table spaces only.

B Full image copy or incremental image copy.

ICType is an input parameter of type VARCHAR(1). The default is **B**.

StatsSchema
Specifies the qualifier for the real-time statistics table names. *StatsSchema* is an input parameter of type VARCHAR(128). The default is **SYSIBM**.

CatlgSchema
Specifies the qualifier for DB2 catalog table names. *CatlgSchema* is an input parameter of type VARCHAR(128). The default is **SYSIBM**.

LocalSchema
Specifies the qualifier for the names of tables that DSNACCOR creates. *LocalSchema* is an input parameter of type VARCHAR(128). The default is **DSNACC**.

ChkLvl
Specifies the types of checking that DSNACCOR performs, and indicates whether to include objects that fail those checks in the DSNACCOR recommendations result set. This value is the sum of any combination of the following values:

0 DSNACCOR performs none of the following actions.

1 For objects that are listed in the recommendations result set, check the SYSTABLESPACE or SYSINDEXES catalog tables to ensure that those objects have not been deleted. If value 16 is **not** also chosen, exclude rows for the deleted objects from the recommendations result set.

DSNACCOR excludes objects from the recommendations result set if those objects are not in the SYSTABLESPACE or SYSINDEXES catalog tables.

When this setting is specified, DSNACCOR does not use EXTENTS>*ExtentLimit* to determine whether a LOB table space should be reorganized.

2 For index spaces that are listed in the recommendations result set, check the SYSTABLES, SYSTABLESPACE, and SYSINDEXES catalog tables to determine the name of the table space that is associated with each index space.

Choosing this value causes DSNACCOR to also check for rows in the recommendations result set for objects that have been deleted but have entries in the real-time statistics tables (value 1). This means that if value 16 is **not** also chosen, rows for deleted objects are excluded from the recommendations result set.

4 Check whether rows that are in the DSNACCOR recommendations result set refer to objects that are in the exception table. For

recommendations result set rows that have corresponding exception table rows, copy the contents of the QUERYTYPE column of the exception table to the INEXCEPTTABLE column of the recommendations result set.

8 Check whether objects that have rows in the recommendations result set are restricted. Indicate the restricted status in the OBJECTSTATUS column of the result set.

16 For objects that are listed in the recommendations result set, check the SYSTABLESPACE or SYSINDEXES catalog tables to ensure that those objects have not been deleted (value 1). In result set rows for deleted objects, specify the word ORPHANED in the OBJECTSTATUS column.

32 Exclude rows from the DSNACCOR recommendations result set for index spaces for which the related table spaces have been recommended for REORG. Choosing this value causes DSNACCOR to perform the actions for values 1 and 2.

64 For index spaces that are listed in the DSNACCOR recommendations result set, check whether the related table spaces are listed in the exception table. For recommendations result set rows that have corresponding exception table rows, copy the contents of the QUERYTYPE column of the exception table to the INEXCEPTTABLE column of the recommendations result set.

ChkLvl is an input parameter of type INTEGER. The default is **7** (values 1+2+4).

Criteria
Narrows the set of objects for which DSNACCOR makes recommendations. This value is the search condition of an SQL WHERE clause. *Criteria* is an input parameter of type VARCHAR(4096). The default is that DSNACCOR makes recommendations for all table spaces and index spaces in the subsystem. The search condition can use any column in the result set and wildcards are allowed.

Restricted
A parameter that is reserved for future use. Specify the null value for this parameter. *Restricted* is an input parameter of type VARCHAR(80).

CRUpdatedPagesPct
Specifies a criterion for recommending a full image copy on a table space or index space. If the following condition is true for a table space, DSNACCOR recommends an image copy:

> The total number of distinct updated pages, divided by the total number of preformatted pages (expressed as a percentage) is greater than *CRUpdatedPagesPct.*

See item 2 in Figure 153 on page 840. If both of the following conditions are true for an index space, DSNACCOR recommends an image copy:

- The total number of distinct updated pages, divided by the total number of preformatted pages (expressed as a percentage) is greater than *CRUpdatedPagesPct.*
- The number of active pages in the index space or partition is greater than *CRIndexSize*. See items 2 and 3 in Figure 154 on page 841.

CRUpdatedPagesPct is an input parameter of type INTEGER. The default is **20**.

CRChangesPct
Specifies a criterion for recommending a full image copy on a table space or index space. If the following condition is true for a table space, DSNACCOR recommends an image copy:

> The total number of insert, update, and delete operations since the last image copy, divided by the total number of rows or LOBs in a table space or partition (expressed as a percentage) is greater than *CRChangesPct.*

See item 3 in Figure 153 on page 840. If both of the following conditions are true for an index table space, DSNACCOR recommends an image copy:

- The total number of insert and delete operations since the last image copy, divided by the total number of entries in the index space or partition (expressed as a percentage) is greater than *CRChangesPct*.
- The number of active pages in the index space or partition is greater than *CRIndexSize.*

See items 2 and 4 in Figure 154 on page 841. *CRChangesPct* is an input parameter of type INTEGER. The default is **10**.

CRDaySncLastCopy
Specifies a criterion for recommending a full image copy on a table space or index space. If the number of days since the last image copy is greater than this value, DSNACCOR recommends an image copy. (See item 1 in Figure 153 on page 840 and item 1 in Figure 154 on page 841.) *CRDaySncLastCopy* is an input parameter of type INTEGER. The default is **7**.

ICRUpdatedPagesPct
Specifies a criterion for recommending an incremental image copy on a table space. If the following condition is true, DSNACCOR recommends an incremental image copy:

> The number of distinct pages that were updated since the last image copy, divided by the total number of active pages in the table space or partition (expressed as a percentage) is greater than *CRUpdatedPagesPct.*

(See item 1 in Figure 155 on page 841.) *ICRUpdatedPagesPct* is an input parameter of type INTEGER. The default is **1**.

ICRChangesPct
Specifies a criterion for recommending an incremental image copy on a table space. If the following condition is true, DSNACCOR recommends an incremental image copy:

> The ratio of the number of insert, update, or delete operations since the last image copy, to the total number of rows or LOBs in a table space or partition (expressed as a percentage) is greater than *ICRChangesPct.*

(See item 2 in Figure 155 on page 841.) *ICRChangesPct* is an input parameter of type INTEGER. The default is **1**.

CRIndexSize
Specifies, when combined with *CRUpdatedPagesPct* or *CRChangesPct*, a criterion for recommending a full image copy on an index space. (See items 2, 3, and 4 in Figure 154 on page 841.) *CRIndexSize* is an input parameter of type INTEGER. The default is **50**.

RRTInsDelUpdPct
Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following condition is true, DSNACCOR recommends running REORG:

The sum of insert, update, and delete operations since the last REORG, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage) is greater than *RRTInsDelUpdPct*

(See item 1 in Figure 156 on page 841.) *RRTInsDelUpdPct* is an input parameter of type INTEGER. The default is **20**.

RRTUnclustInsPct
Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following condition is true, DSNACCOR recommends running REORG:

The number of unclustered insert operations, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage) is greater than *RRTUnclustInsPct*.

(See item 2 in Figure 156 on page 841.) *RRTUnclustInsPct* is an input parameter of type INTEGER. The default is **10**.

RRTDisorgLOBPct
Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following condition is true, DSNACCOR recommends running REORG:

The number of imperfectly chunked LOBs, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage) is greater than *RRTDisorgLOBPct*.

(See item 3 in Figure 156 on page 841.) *RRTDisorgLOBPct* is an input parameter of type INTEGER. The default is **10**.

RRTMassDelLimit
Specifies a criterion for recommending that the REORG utility is to be run on a table space. If one of the following values is greater than *RRTMassDelLimit*, DSNACCOR recommends running REORG:

- The number of mass deletes from a segmented or LOB table space since the last REORG or LOAD REPLACE
- The number of dropped tables from a nonsegmented table space since the last REORG or LOAD REPLACE

(See item 5 in Figure 156 on page 841.) *RRTMassDelLimit* is an input parameter of type INTEGER. The default is **0**.

RRTIndRefLimit
Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following value is greater than *RRTIndRefLimit*, DSNACCOR recommends running REORG:

The total number of overflow records that were created since the last REORG or LOAD REPLACE, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage)

(See item 4 in Figure 156 on page 841.) *RRTIndRefLimit* is an input parameter of type INTEGER. The default is **10**.

RRIInsertDeletePct
Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRIInsertDeletePct*, DSNACCOR recommends running REORG:

The sum of the number of index entries that were inserted and deleted since the last REORG, divided by the total number of index entries in the index space or partition (expressed as a percentage)

(See item 1 in Figure 157 on page 842.) This is an input parameter of type INTEGER. The default is **20**.

RRIAppendInsertPct
Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRIAppendInsertPct*, DSNACCOR recommends running REORG:

The number of index entries that were inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE with a key value greater than the maximum key value in the index space or partition, divided by the number of index entries in the index space or partition (expressed as a percentage)

(See item 2 in Figure 157 on page 842.) *RRIInsertDeletePct* is an input parameter of type INTEGER. The default is **10**.

RRIPseudoDeletePct
Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRIPseudoDeletePct*, DSNACCOR recommends running REORG:

The number of index entries that were pseudo-deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE, divided by the number of index entries in the index space or partition (expressed as a percentage)

(See item 3 in Figure 157 on page 842.) *RRIPseudoDeletePct* is an input parameter of type INTEGER. The default is **10**.

RRIMassDelLimit
Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the number of mass deletes from an index space or partition since the last REORG, REBUILD, or LOAD REPLACE is greater than this value, DSNACCOR recommends running REORG.

(See item 4 in Figure 157 on page 842.) *RRIMassDelLimit* is an input parameter of type INTEGER. The default is **0**.

RRILeafLimit
Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRILeafLimit*, DSNACCOR recommends running REORG:

The number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was far from the location of the original page, divided by the total number of active pages in the index space or partition (expressed as a percentage)

(See item 5 in Figure 157 on page 842.) *RRILeafLimit* is an input parameter of type INTEGER. The default is **10**.

RRINumLevelsLimit
Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRINumLevelsLimit*, DSNACCOR recommends running REORG:

The number of levels in the index tree that were added or removed since the last REORG, REBUILD INDEX, or LOAD REPLACE

(See item 6 in Figure 157 on page 842.) *RRINumLevelsLimit* is an input parameter of type INTEGER. The default is **0**.

SRTInsDelUpdPct
Specifies, when combined with *SRTInsDelUpdAbs*, a criterion for

recommending that the RUNSTATS utility is to be run on a table space. If both of the following conditions are true, DSNACCOR recommends running RUNSTATS:

- The number of insert, update, or delete operations since the last RUNSTATS on a table space or partition, divided by the total number of rows or LOBs in table space or partition (expressed as a percentage) is greater than *SRTInsDelUpdPct.*
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRTInsDelUpdAbs.*

(See items 1 and 2 in Figure 158 on page 842.) *SRTInsDelUpdPct* is an input parameter of type INTEGER. The default is **20**.

SRTInsDelUpdAbs
Specifies, when combined with *SRTInsDelUpdPct*, a criterion for recommending that the RUNSTATS utility is to be run on a table space. If both of the following conditions are true, DSNACCOR recommends running RUNSTATS:

- The number of insert, update, and delete operations since the last RUNSTATS on a table space or partition, divided by the total number of rows or LOBs in table space or partition (expressed as a percentage) is greater than *SRTInsDelUpdPct.*
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRTInsDelUpdAbs.*

(See items 1 and 2 in Figure 158 on page 842.) *SRTInsDelUpdAbs* is an input parameter of type INTEGER. The default is **0**.

SRTMassDelLimit
Specifies a criterion for recommending that the RUNSTATS utility is to be run on a table space. If the following condition is true, DSNACCOR recommends running RUNSTATS:

- The number of mass deletes from a table space or partition since the last REORG or LOAD REPLACE is greater than *SRTMassDelLimit.*

(See item 3 in Figure 158 on page 842.) *SRTMassDelLimit* is an input parameter of type INTEGER. The default is **0**.

SRIInsDelPct
Specifies, when combined with *SRIInsDelAbs*, a criterion for recommending that the RUNSTATS utility is to be run on an index space. If both of the following conditions are true, DSNACCOR recommends running RUNSTATS:

- The number of inserted and deleted index entries since the last RUNSTATS on an index space or partition, divided by the total number of index entries in the index space or partition (expressed as a percentage) is greater than *SRIInsDelUpdPct.*
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRIInsDelUpdAbs.*

(See items 1 and 2 in Figure 159 on page 842.) *SRIInsDelPct* is an input parameter of type INTEGER. The default is **20**.

SRIInsDelAbs
Specifies, when combined with *SRIInsDelPct*, specifies a criterion for recommending that the RUNSTATS utility is to be run on an index space. If the following condition is true, DSNACCOR recommends running RUNSTATS:

- The number of inserted and deleted index entries since the last RUNSTATS on an index space or partition, divided by the total number of index entries in the index space or partition (expressed as a percentage) is greater than *SRIInsDelUpdPct.*
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRIInsDelUpdAbs*,

(See items 1 and 2 in Figure 159 on page 842.) *SRIInsDelAbs* is an input parameter of type INTEGER. The default is **0**.

SRIMassDelLimit
Specifies a criterion for recommending that the RUNSTATS utility is to be run on an index space. If the number of mass deletes from an index space or partition since the last REORG, REBUILD INDEX, or LOAD REPLACE is greater than this value, DSNACCOR recommends running RUNSTATS.

(See item 3 in Figure 159 on page 842.) *SRIMassDelLimit* is an input parameter of type INTEGER. The **default** is **0**.

ExtentLimit
Specifies a criterion for recommending that the REORG utility is to be run on a table space or index space. Also specifies that DSNACCOR is to warn the user that the table space or index space has used too many extents. DSNACCOR recommends running REORG, and altering data set allocations if the following condition is true:

- The number of physical extents in the index space, table space, or partition is greater than *ExtentLimit.*

(See Figure 160 on page 842.) *ExtentLimit* is an input parameter of type INTEGER. The default is **50**.

LastStatement
When DSNACCOR returns a severe error (return code 12), this field contains the SQL statement that was executing when the error occurred. *LastStatement* is an output parameter of type VARCHAR(8012).

ReturnCode
The return code from DSNACCOR execution. Possible values are:

0 DSNACCOR executed successfully. The *ErrorMsg* parameter contains the approximate percentage of the total number of objects in the subsystem that have information in the real-time statistics tables.

4 DSNACCOR completed, but one or more input parameters might be incompatible. The *ErrorMsg* parameter contains the input parameters that might be incompatible.

8 DSNACCOR terminated with errors. The *ErrorMsg* parameter contains a message that describes the error.

12 DSNACCOR terminated with severe errors. The *ErrorMsg* parameter contains a message that describes the error. The *LastStatement* parameter contains the SQL statement that was executing when the error occurred.

14 DSNACCOR terminated because it could not access one or more of the real-time statistics tables. The *ErrorMsg* parameter contains the names of the tables that DSNACCOR could not access.

15 DSNACCOR terminated because it encountered a problem with one of the declared temporary tables that it defines and uses.

16 DSNACCOR terminated because it could not define a declared temporary table. No table spaces were defined in the TEMP database.

NULL DSNACCOR terminated but could not set a return code.

ReturnCode is an output parameter of type INTEGER.

ErrorMsg
Contains information about DSNACCOR execution. If DSNACCOR runs successfully (*ReturnCode*=0), this field contains the approximate percentage of objects in the subsystem that are in the real-time statistics tables. Otherwise, this field contains error messages. *ErrorMsg* is an output parameter of type VARCHAR(1331).

IFCARetCode
Contains the return code from an IFI COMMAND call. DSNACCOR issues commands through the IFI interface to determine the status of objects. *IFCARetCode* is an output parameter of type INTEGER.

IFCAResCode
Contains the reason code from an IFI COMMAND call. *IFCAResCode* is an output parameter of type INTEGER.

ExcessBytes
Contains the number of bytes of information that did not fit in the IFI return area after an IFI COMMAND call. *ExcessBytes* is an output parameter of type INTEGER.

DSNACCOR formulas for recommending actions

The following formulas specify the criteria that DSNACCOR uses for its recommendations and warnings. The variables in italics are DSNACCOR input parameters. The capitalized variables are columns of the SYSIBM.TABLESPACESTATS or SYSIBM.INDEXSPACESTATS tables. The numbers to the right of selected items are reference numbers for the option descriptions in "DSNACCOR option descriptions" on page 832.

Figure 153 shows the formula that DSNACCOR uses to recommend a full image copy on a table space.

```
((QueryType='COPY' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL') AND
 ICType='F') AND
 (COPYLASTTIME IS NULL OR
 REORGLASTTIME>COPYLASTTIME OR
 LOADRLASTTIME>COPYLASTTIME OR
 (CURRENT DATE-COPYLASTTIME)>CRDaySncLastCopy OR      1
 (COPYUPDATEDPAGES*100)/NACTIVE>CRUpdatedPagesPct OR  2
 (COPYCHANGES*100)/TOTALROWS>CRChangesPct)            3
```

Figure 153. DSNACCOR formula for recommending a full image copy on a table space

Figure 154 on page 841 shows the formula that DSNACCOR uses to recommend a full image copy on an index space.

```
((QueryType='COPY' OR QueryType='ALL') AND
 (ObjectType='IX' OR ObjectType='ALL') AND
 (ICType='F'  OR ICType='B')) AND
 (COPYLASTTIME IS NULL OR
 REORGLASTTIME>COPYLASTTIME OR
 LOADRLASTTIME>COPYLASTTIME OR
 REBUILDLASTTIME>COPYLASTTIME OR
 (CURRENT DATE-COPYLASTTIME)>CRDaySncLastCopy OR       1
 (NACTIVE>CRIndexSize AND                              2
 ((COPYUPDATEDPAGES*100)/NACTIVE>CRUpdatedPagesPct OR  3
 (COPYCHANGES*100)/TOTALENTRIES>CRChangesPct)))        4
```

Figure 154. DSNACCOR formula for recommending a full image copy on an index space

Figure 155 shows the formula that DSNACCOR uses to recommend an incremental image copy on a table space.

```
((QueryType='COPY' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL') AND
 ICType='I' AND
 COPYLASTTIME IS NOT NULL) AND
 (LOADRLASTTIME>COPYLASTTIME OR
 REORGLASTTIME>COPYLASTTIME OR
 (COPYUPDATEDPAGES*100)/NACTIVE>ICRUpdatedPagesPct OR  1
 (COPYCHANGES*100)/TOTALROWS>ICRChangesPct))           2
```

Figure 155. DSNACCOR formula for recommending an incremental image copy on a table space

Figure 156 shows the formula that DSNACCOR uses to recommend a REORG on a table space. If the table space is a LOB table space, and CHCKLVL=1, the formula does not include EXTENTS>*ExtentLimit*.

```
((QueryType='REORG' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL')) AND
 (REORGLASTTIME IS NULL OR
 ((REORGINSERTS+REORGDELETES+REORGUPDATES)*100)/TOTALROWS>RRTInsDelUpdPct OR  1
 (REORGUNCLUSTINS*100)/TOTALROWS>RRTUnclustInsPct OR                          2
 (REORGDISORGLOB*100)/TOTALROWS>RRTDisorgLOBPct OR                            3
 ((REORGNEARINDREF+REORGFARINDREF)*100)/TOTALROWS>RRTIndRefLimit OR           4
 REORGMASSDELETE>RRTMassDelLimit OR                                           5
 EXTENTS>ExtentLimit)                                                         6
```

Figure 156. DSNACCOR formula for recommending a REORG on a table space

Figure 157 on page 842 shows the formula that DSNACCOR uses to recommend a REORG on an index space.

```
((QueryType='REORG' OR QueryType='ALL') AND
 (ObjectType='IX' OR ObjectType='ALL')) AND
 (REORGLASTTIME IS NULL OR
 ((REORGINSERTS+REORGDELETES)*100)/TOTALENTRIES>RRIInsertDeletePct OR          1
 (REORGAPPENDINSERT*100)/TOTALENTRIES>RRIAppendInsertPct OR                    2
 (REORGPSEUDODELETES*100)/TOTALENTRIES>RRIPseudoDeletePct OR                   3
 REORGMASSDELETE>RRIMassDeleteLimit OR                                         4
 (REORGLEAFFAR*100)/NACTIVE>RRILeafLimit OR                                    5
 REORGNUMLEVELS>RRINumLevelsLimit OR                                           6
 EXTENTS>ExtentLimit)                                                          7
```

Figure 157. DSNACCOR formula for recommending a REORG on an index space

Figure 158 shows the formula that DSNACCOR uses to recommend RUNSTATS on a table space.

```
((QueryType='RUNSTATS' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL')) AND
 (STATSLASTTIME IS NULL OR
 (((STATSINSERTS+STATSDELETES+STATSUPDATES)*100)/TOTALROWS>SRTInsDelUpdPct AND  1
 (STATSINSERTS+STATSDELETES+STATSUPDATES)>SRTInsDelUpdAbs) OR                    2
 STATSMASSDELETE>SRTMassDeleteLimit)                                             3
```

Figure 158. DSNACCOR formula for recommending RUNSTATS on a table space

Figure 159 shows the formula that DSNACCOR uses to recommend RUNSTATS on an index space.

```
((QueryType='RUNSTATS' OR QueryType='ALL') AND
 (ObjectType='IX' OR ObjectType='ALL')) AND
 (STATSLASTTIME IS NULL OR
 (((STATSINSERTS+STATSDELETES)*100)/TOTALENTRIES>SRIInsDelUpdPct AND   1
 (STATSINSERTS+STATSDELETES)>SRIInsDelPct) OR                          2
 STATSMASSDELETE>SRIInsDelAbs)                                         3
```

Figure 159. DSNACCOR formula for recommending RUNSTATS on an index space

Figure 160 shows the formula that DSNACCOR uses to that too many index space or table space extents have been used.

```
EXTENTS>ExtentLimit
```

Figure 160. DSNACCOR formula for warning that too many data set extents for a table space or index space are used

Using an exception table

An exception table is an optional, user-created DB2 table that you can use to place information in the INEXCEPTTABLE column of the recommendations result set. You can put any information in the INEXCEPTTABLE column, but the most common use of this column is to filter the recommendations result set. Each row in the exception table represents an object for which you want to provide information for the recommendations result set.

To create the exception table, execute a CREATE TABLE statement similar to the following one. You can include other columns in the exception table, but you must include at least the columns that are shown.

```
CREATE TABLE DSNACC.EXCEPT_TBL
 (DBNAME CHAR(8) NOT NULL,
  NAME CHAR(8) NOT NULL,
  QUERYTYPE CHAR(40))
 CCSID EBCDIC;
```

The meanings of the columns are:

DBNAME
The database name for an object in the exception table.

NAME
The table space name or index space name for an object in the exception table.

QUERYTYPE
The information that you want to place in the INEXCEPTTABLE column of the recommendations result set.

If you put a null value in this column, DSNACCOR puts the value YES in the INEXCEPTTABLE column of the recommendations result set row for the object that matches the DBNAME and NAME values.

Recommendation: If you plan to put many rows in the exception table, create a nonunique index on DBNAME, NAME, and QUERYTYPE.

After you create the exception table, insert a row for each object for which you want to include information in the INEXCEPTTABLE column.

Example: Suppose that you want the INEXCEPTTABLE column to contain the string 'IRRELEVANT' for table space STAFF in database DSNDB04. You also want the INEXCEPTTABLE column to contain 'CURRENT' for table space DSN8S81D in database DSN8D81A. Execute these INSERT statements:

```
INSERT INTO DSNACC.EXCEPT_TBL VALUES('DSNDB04 ', 'STAFF ', 'IRRELEVANT');
INSERT INTO DSNACC.EXCEPT_TBL VALUES('DSN8D81A', 'DSN8S81D', 'CURRENT');
```

To use the contents of INEXCEPTTABLE for filtering, include a condition that involves the INEXCEPTTABLE column in the search condition that you specify in your *Criteria* input parameter.

Example: Suppose that you want to include all rows for database DSNDB04 in the recommendations result set, except for those rows that contain the string 'IRRELEVANT' in the INEXCEPTTABLE column. You might include the following search condition in your *Criteria* input parameter:

```
DBNAME='DSNDB04' AND INEXCEPTTABLE<>'IRRELEVANT'
```

Example of DSNACCOR invocation

Figure 161 on page 844 is a COBOL example that shows variable declarations and an SQL CALL for obtaining recommendations for objects in databases DSN8D81A and DSN8D81L. This example also outlines the steps that you need to perform to retrieve the two result sets that DSNACCOR returns. These result sets are described in "DSNACCOR output" on page 847. See *DB2 Application Programming and SQL Guide* for more information about how to retrieve result sets from a stored procedure.

```
 WORKING-STORAGE SECTION.
:
**********************
* DSNACCOR PARAMETERS *
**********************
 01  QUERYTYPE.
     49   QUERYTYPE-LN        PICTURE S9(4) COMP VALUE 40.
     49   QUERYTYPE-DTA       PICTURE X(40)  VALUE 'ALL'.
 01  OBJECTTYPE.
     49   OBJECTTYPE-LN       PICTURE S9(4) COMP VALUE 3.
     49   OBJECTTYPE-DTA      PICTURE X(3)  VALUE 'ALL'.
 01  ICTYPE.
     49   ICTYPE-LN           PICTURE S9(4) COMP VALUE 1.
     49   ICTYPE-DTA          PICTURE X(1)  VALUE 'B'.
 01  STATSSCHEMA.
     49   STATSSCHEMA-LN      PICTURE S9(4) COMP VALUE 128.
     49   STATSSCHEMA-DTA     PICTURE X(128)  VALUE 'SYSIBM'.
 01  CATLGSCHEMA.
 49   CATLGSCHEMA-LN      PICTURE S9(4) COMP VALUE 128.
     49   CATLGSCHEMA-DTA     PICTURE X(128)  VALUE 'SYSIBM'.
 01  LOCALSCHEMA.
     49   LOCALSCHEMA-LN      PICTURE S9(4) COMP VALUE 128.
     49   LOCALSCHEMA-DTA     PICTURE X(128)  VALUE 'DSNACC'.
 01  CHKLVL                   PICTURE S9(9) COMP VALUE +3.
 01  CRITERIA.
     49   CRITERIA-LN         PICTURE S9(4) COMP VALUE 4096.
     49   CRITERIA-DTA        PICTURE X(4096)  VALUE SPACES.
 01  RESTRICTED.
     49   RESTRICTED-LN       PICTURE S9(4) COMP VALUE 80.
     49   RESTRICTED-DTA      PICTURE X(80)  VALUE SPACES.
 01  CRUPDATEDPAGESPCT        PICTURE S9(9) COMP VALUE +0.
 01  CRCHANGESPCT             PICTURE S9(9) COMP VALUE +0.
 01  CRDAYSNCLASTCOPY         PICTURE S9(9) COMP VALUE +0.
 01  ICRUPDATEDPAGESPCT       PICTURE S9(9) COMP VALUE +0.
 01  ICRCHANGESPCT            PICTURE S9(9) COMP VALUE +0.
 01  CRINDEXSIZE              PICTURE S9(9) COMP VALUE +0.
 01  RRTINSDELUPDPCT          PICTURE S9(9) COMP VALUE +0.
 01  RRTUNCLUSTINSPCT         PICTURE S9(9) COMP VALUE +0.
 01  RRTDISORGLOBPCT          PICTURE S9(9) COMP VALUE +0.
 01  RRTMASSDELLIMIT          PICTURE S9(9) COMP VALUE +0.
 01  RRTINDREFLIMIT           PICTURE S9(9) COMP VALUE +0.
 01  RRIINSERTDELETEPCT       PICTURE S9(9) COMP VALUE +0.
 01  RRIAPPENDINSERTPCT       PICTURE S9(9) COMP VALUE +0.
 01  RRIPSEUDODELETEPCT       PICTURE S9(9) COMP VALUE +0.
 01  RRIMASSDELLIMIT          PICTURE S9(9) COMP VALUE +0.
 01  RRILEAFLIMIT             PICTURE S9(9) COMP VALUE +0.
 01  RRINUMLEVELSLIMIT        PICTURE S9(9) COMP VALUE +0.
 01  SRTINSDELUPDPCT          PICTURE S9(9) COMP VALUE +0.
 01  SRTINSDELUPDABS          PICTURE S9(9) COMP VALUE +0.
 01  SRTMASSDELLIMIT          PICTURE S9(9) COMP VALUE +0.
 01  SRIINSDELUPDPCT          PICTURE S9(9) COMP VALUE +0.
 01  SRIINSDELUPDABS          PICTURE S9(9) COMP VALUE +0.
 01  SRIMASSDELLIMIT          PICTURE S9(9) COMP VALUE +0.
 01  EXTENTLIMIT              PICTURE S9(9) COMP VALUE +0.
 01  LASTSTATEMENT.
     49   LASTSTATEMENT-LN    PICTURE S9(4) COMP VALUE 8012.
     49   LASTSTATEMENT-DTA   PICTURE X(8012)  VALUE SPACES.
 01  RETURNCODE               PICTURE S9(9) COMP VALUE +0.
 01  ERRORMSG.
     49   ERRORMSG-LN         PICTURE S9(4) COMP VALUE 1331.
     49   ERRORMSG-DTA        PICTURE X(1331)  VALUE SPACES.
 01  IFCARETCODE              PICTURE S9(9) COMP VALUE +0.
 01  IFCARESCODE              PICTURE S9(9) COMP VALUE +0.
 01  EXCESSBYTES              PICTURE S9(9) COMP VALUE +0.
```

Figure 161. Example of DSNACCOR invocation (Part 1 of 4)

```
****************************************
* INDICATOR VARIABLES.                 *
* INITIALIZE ALL NON-ESSENTIAL INPUT   *
* VARIABLES TO -1, TO INDICATE THAT THE *
* INPUT VALUE IS NULL.                 *
****************************************
 01  QUERYTYPE-IND            PICTURE S9(4) COMP-4 VALUE +0.
 01  OBJECTTYPE-IND           PICTURE S9(4) COMP-4 VALUE +0.
 01  ICTYPE-IND               PICTURE S9(4) COMP-4 VALUE +0.
 01  STATSSCHEMA-IND          PICTURE S9(4) COMP-4 VALUE -1.
 01  CATLGSCHEMA-IND          PICTURE S9(4) COMP-4 VALUE -1.
 01  LOCALSCHEMA-IND          PICTURE S9(4) COMP-4 VALUE -1.
 01  CHKLVL-IND               PICTURE S9(4) COMP-4 VALUE -1.
 01  CRITERIA-IND             PICTURE S9(4) COMP-4 VALUE -1.
 01  RESTRICTED-IND           PICTURE S9(4) COMP-4 VALUE -1.
 01  CRUPDATEDPAGESPCT-IND    PICTURE S9(4) COMP-4 VALUE -1.
 01  CRCHANGESPCT-IND         PICTURE S9(4) COMP-4 VALUE -1.
 01  CRDAYSNCLASTCOPY-IND     PICTURE S9(4) COMP-4 VALUE -1.
 01  ICRUPDATEDPAGESPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
 01  ICRCHANGESPCT-IND        PICTURE S9(4) COMP-4 VALUE -1.
 01  CRINDEXSIZE-IND          PICTURE S9(4) COMP-4 VALUE -1.
 01  RRTINSDELUPDPCT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  RRTUNCLUSTINSPCT-IND     PICTURE S9(4) COMP-4 VALUE -1.
 01  RRTDISORGLOBPCT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  RRTMASSDELLIMIT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  RRTINDREFLIMIT-IND       PICTURE S9(4) COMP-4 VALUE -1.
 01  RRIINSERTDELETEPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
 01  RRIAPPENDINSERTPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
 01  RRIPSEUDODELETEPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
 01  RRIMASSDELLIMIT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  RRILEAFLIMIT-IND         PICTURE S9(4) COMP-4 VALUE -1.
 01  RRINUMLEVELSLIMIT-IND    PICTURE S9(4) COMP-4 VALUE -1.
 01  SRTINSDELUPDPCT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRTINSDELUPDABS-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRTMASSDELLIMIT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRIINSDELUPDPCT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRIINSDELUPDABS-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRIMASSDELLIMIT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  EXTENTLIMIT-IND          PICTURE S9(4) COMP-4 VALUE -1.
 01  LASTSTATEMENT-IND        PICTURE S9(4) COMP-4 VALUE +0.
 01  RETURNCODE-IND           PICTURE S9(4) COMP-4 VALUE +0.
 01  ERRORMSG-IND             PICTURE S9(4) COMP-4 VALUE +0.
 01  IFCARETCODE-IND          PICTURE S9(4) COMP-4 VALUE +0.
 01  IFCARESCODE-IND          PICTURE S9(4) COMP-4 VALUE +0.
 01  EXCESSBYTES-IND          PICTURE S9(4) COMP-4 VALUE +0.

 PROCEDURE DIVISION.
:
:
*********************************************************
* SET VALUES FOR DSNACCOR INPUT PARAMETERS:             *
* - USE THE CHKLVL PARAMETER TO CAUSE DSNACCOR TO CHECK *
*   FOR ORPHANED OBJECTS AND INDEX SPACES WITHOUT       *
*   TABLE SPACES, BUT INCLUDE THOSE OBJECTS IN THE      *
*   RECOMMENDATIONS RESULT SET (CHKLVL=1+2+16=19)       *
* - USE THE CRITERIA PARAMETER TO CAUSE DSNACCOR TO     *
*   MAKE RECOMMENDATIONS ONLY FOR OBJECTS IN DATABASES  *
*   DSN8D81A AND DSN8D81L.                              *
```

Figure 161. Example of DSNACCOR invocation (Part 2 of 4)

```
* - FOR THE FOLLOWING PARAMETERS, SET THESE VALUES,     *
*   WHICH ARE LOWER THAN THE DEFAULTS:                  *
*   CRUPDATEDPAGESPCT   4                               *
*   CRCHANGESPCT        2                               *
*   RRTINSDELUPDPCT     2                               *
*   RRTUNCLUSTINSPCT    5                               *
*   RRTDISORGLOBPCT     5                               *
*   RRIAPPENDINSERTPCT  5                               *
*   SRTINSDELUPDPCT     5                               *
*   SRIINSDELUPDPCT     5                               *
*   EXTENTLIMIT         3                               *
*******************************************************
     MOVE 19 TO CHKLVL.
     MOVE SPACES TO CRITERIA-DTA.
     MOVE 'DBNAME = ''DSN8D81A'' OR DBNAME = ''DSN8D81L'''
          TO CRITERIA-DTA.
     MOVE 46 TO CRITERIA-LN.
     MOVE 4 TO CRUPDATEDPAGESPCT.
     MOVE 2 TO CRCHANGESPCT.
     MOVE 2 TO RRTINSDELUPDPCT.
     MOVE 5 TO RRTUNCLUSTINSPCT.
     MOVE 5 TO RRTDISORGLOBPCT.
     MOVE 5 TO RRIAPPENDINSERTPCT.
     MOVE 5 TO SRTINSDELUPDPCT.
     MOVE 5 TO SRIINSDELUPDPCT.
     MOVE 3 TO EXTENTLIMIT.
*********************************
* INITIALIZE OUTPUT PARAMETERS *
*********************************
     MOVE SPACES TO LASTSTATEMENT-DTA.
     MOVE 1 TO LASTSTATEMENT-LN.
     MOVE 0 TO RETURNCODE-O2.
     MOVE SPACES TO ERRORMSG-DTA.
     MOVE 1 TO ERRORMSG-LN.
     MOVE 0 TO IFCARETCODE.
     MOVE 0 TO IFCARESCODE.
     MOVE 0 TO EXCESSBYTES.
*******************************************************
* SET THE INDICATOR VARIABLES TO 0 FOR NON-NULL INPUT *
* PARAMETERS (PARAMETERS FOR WHICH YOU DO NOT WANT    *
* DSNACCOR TO USE DEFAULT VALUES) AND FOR OUTPUT      *
* PARAMETERS.                                         *
*******************************************************
     MOVE 0 TO CHKLVL-IND.
     MOVE 0 TO CRITERIA-IND.
     MOVE 0 TO CRUPDATEDPAGESPCT-IND.
     MOVE 0 TO CRCHANGESPCT-IND.
     MOVE 0 TO RRTINSDELUPDPCT-IND.
     MOVE 0 TO RRTUNCLUSTINSPCT-IND.
     MOVE 0 TO RRTDISORGLOBPCT-IND.
     MOVE 0 TO RRIAPPENDINSERTPCT-IND.
     MOVE 0 TO SRTINSDELUPDPCT-IND.
     MOVE 0 TO SRIINSDELUPDPCT-IND.
     MOVE 0 TO EXTENTLIMIT-IND.
     MOVE 0 TO LASTSTATEMENT-IND.
     MOVE 0 TO RETURNCODE-IND.
     MOVE 0 TO ERRORMSG-IND.
     MOVE 0 TO IFCARETCODE-IND.
     MOVE 0 TO IFCARESCODE-IND.
     MOVE 0 TO EXCESSBYTES-IND.
⋮
```

Figure 161. Example of DSNACCOR invocation (Part 3 of 4)

```
*****************
* CALL DSNACCOR *
*****************
     EXEC SQL
      CALL SYSPROC.DSNACCOR
      (:QUERYTYPE           :QUERYTYPE-IND,
       :OBJECTTYPE          :OBJECTTYPE-IND,
       :ICTYPE              :ICTYPE-IND,
       :STATSSCHEMA         :STATSSCHEMA-IND,
       :CATLGSCHEMA         :CATLGSCHEMA-IND,
       :LOCALSCHEMA         :LOCALSCHEMA-IND,
       :CHKLVL              :CHKLVL-IND,
       :CRITERIA            :CRITERIA-IND,
       :RESTRICTED          :RESTRICTED-IND,
       :CRUPDATEDPAGESPCT   :CRUPDATEDPAGESPCT-IND,
       :CRCHANGESPCT        :CRCHANGESPCT-IND,
       :CRDAYSNCLASTCOPY    :CRDAYSNCLASTCOPY-IND,
       :ICRUPDATEDPAGESPCT  :ICRUPDATEDPAGESPCT-IND,
       :ICRCHANGESPCT       :ICRCHANGESPCT-IND,
       :CRINDEXSIZE         :CRINDEXSIZE-IND,
       :RRTINSDELUPDPCT     :RRTINSDELUPDPCT-IND,
       :RRTUNCLUSTINSPCT    :RRTUNCLUSTINSPCT-IND,
       :RRTDISORGLOBPCT     :RRTDISORGLOBPCT-IND,
       :RRTMASSDELLIMIT     :RRTMASSDELLIMIT-IND,
       :RRTINDREFLIMIT      :RRTINDREFLIMIT-IND,
       :RRIINSERTDELETEPCT  :RRIINSERTDELETEPCT-IND,
       :RRIAPPENDINSERTPCT  :RRIAPPENDINSERTPCT-IND,
       :RRIPSEUDODELETEPCT  :RRIPSEUDODELETEPCT-IND,
       :RRIMASSDELLIMIT     :RRIMASSDELLIMIT-IND,
       :RRILEAFLIMIT        :RRILEAFLIMIT-IND,
       :RRINUMLEVELSLIMIT   :RRINUMLEVELSLIMIT-IND,
       :SRTINSDELUPDPCT     :SRTINSDELUPDPCT-IND,
       :SRTINSDELUPDABS     :SRTINSDELUPDABS-IND,
       :SRTMASSDELLIMIT     :SRTMASSDELLIMIT-IND,
       :SRIINSDELUPDPCT     :SRIINSDELUPDPCT-IND,
       :SRIINSDELUPDABS     :SRIINSDELUPDABS-IND,
       :SRIMASSDELLIMIT     :SRIMASSDELLIMIT-IND,
       :EXTENTLIMIT         :EXTENTLIMIT-IND,
       :LASTSTATEMENT       :LASTSTATEMENT-IND,
       :RETURNCODE          :RETURNCODE-IND,
       :ERRORMSG            :ERRORMSG-IND,
       :IFCARETCODE         :IFCARETCODE-IND,
       :IFCARESCODE         :IFCARESCODE-IND,
       :EXCESSBYTES         :EXCESSBYTES-IND)
     END-EXEC.
************************************************************
* ASSUME THAT THE SQL CALL RETURNED +466, WHICH MEANS THAT  *
* RESULT SETS WERE RETURNED. RETRIEVE RESULT SETS.          *
************************************************************
* LINK EACH RESULT SET TO A LOCATOR VARIABLE
     EXEC SQL ASSOCIATE LOCATORS (:LOC1, :LOC2)
       WITH PROCEDURE SYSPROC.DSNACCOR
     END-EXEC.
* LINK A CURSOR TO EACH RESULT SET
     EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :LOC1
     END-EXEC.
     EXEC SQL ALLOCATE C2 CURSOR FOR RESULT SET :LOC2
     END-EXEC.
* PERFORM FETCHES USING C1 TO RETRIEVE ALL ROWS FROM FIRST RESULT SET
* PERFORM FETCHES USING C2 TO RETRIEVE ALL ROWS FROM SECOND RESULT SET
```

Figure 161. Example of DSNACCOR invocation (Part 4 of 4)

DSNACCOR output

If DSNACCOR executes successfully, in addition to the output parameters described in "DSNACCOR option descriptions" on page 832, DSNACCOR returns two result sets.

The first result set contains the results from IFI COMMAND calls that DSNACCOR makes. Table 162 on page 848 shows the format of the first result set.

Table 162. Result set row for first DSNACCOR result set

Column name	Data type	Contents
RS_SEQUENCE	INTEGER	Sequence number of the output line
RS_DATA	CHAR(80)	A line of command output

The second result set contains DSNACCOR's recommendations. This result set contains one or more rows for a table space or index space. A nonpartitioned table space or nonpartitioning index space can have at most one row in the result set. A partitioned table space or partitioning index space can have at most one row for each partition. A table space, index space, or partition has a row in the result set if both of the following conditions are true:

- If the *Criteria* input parameter contains a search condition, the search condition is true for the table space, index space, or partition.
- DSNACCOR recommends at least one action for the table space, index space, or partition.

Table 163 shows the columns of a result set row.

Table 163. Result set row for second DSNACCOR result set

Column name	Data type	Description
DBNAME	CHAR(8)	Name of the database that contains the object.
NAME	CHAR(8)	Table space or index space name.
PARTITION	INTEGER	Data set number or partition number.
OBJECTTYPE	CHAR(2)	DB2 object type: • TS for a table space • IX for an index space
OBJECTSTATUS	CHAR(36)	Status of the object: • ORPHANED, if the object is an index space with no corresponding table space, or if the object does not exist • If the object is in a restricted state, one of the following values: – TS=*restricted-state*, if OBJECTTYPE is TS – IX=*restricted-state*, if OBJECTTYPE is IX *restricted-state* is one of the status codes that appear in DISPLAY DATABASE output. See Chapter 2 of *DB2 Command Reference* for details. • A, if the object is in an advisory state. • L, if the object is a logical partition, but not in an advisory state. • AL, if the object is a logical partition and in an advisory state.
IMAGECOPY	CHAR(3)	COPY recommendation: • If OBJECTTYPE is TS: FUL (full image copy), INC (incremental image copy), or NO • If OBJECTTYPE is IX: YES or NO
RUNSTATS	CHAR(3)	RUNSTATS recommendation: YES or NO.
EXTENTS	CHAR(3)	Indicates whether the data sets for the object have exceeded *ExtentLimit*: YES or NO.
REORG	CHAR(3)	REORG recommendation: YES or NO.

Table 163. Result set row for second DSNACCOR result set (continued)

Column name	Data type	Description
INEXCEPTTABLE	CHAR(40)	A string that contains one of the following values: • Text that you specify in the QUERYTYPE column of the exception table. • YES, if you put a row in the exception table for the object that this result set row represents, but you specify NULL in the QUERYTYPE column. • NO, if the exception table exists but does not have a row for the object that this result set row represents. • Null, if the exception table does not exist, or if the *ChkLvl* input parameter does not include the value 4.
ASSOCIATEDTS	CHAR(8)	If OBJECTTYPE is IX and the *ChkLvl* input parameter includes the value 2, this value is the name of the table space that is associated with the index space. Otherwise null.
COPYLASTTIME	TIMESTAMP	Timestamp of the last full image copy on the object. Null if COPY was never run, or if the last COPY execution was terminated.
LOADRLASTTIME	TIMESTAMP	Timestamp of the last LOAD REPLACE on the object. Null if LOAD REPLACE was never run, or if the last LOAD REPLACE execution was terminated.
REBUILDLASTTIME	TIMESTAMP	Timestamp of the last REBUILD INDEX on the object. Null if REBUILD INDEX was never run, or if the last REBUILD INDEX execution was terminated.
CRUPDPGSPCT	INTEGER	If OBJECTTYPE is TS or IX and IMAGECOPY is YES, the ratio of distinct updated pages to preformatted pages, expressed as a percentage. Otherwise null.
CRCPYCHGPCT	INTEGER	If OBJECTTYPE is TS and IMAGECOPY is YES, the ratio of the total number insert, update, and delete operations since the last image copy to the total number of rows or LOBs in the table space or partition, expressed as a percentage. If OBJECTTYPE is IX and IMAGECOPY is YES, the ratio of the total number of insert and delete operations since the last image copy to the total number of entries in the index space or partition, expressed as a percentage. Otherwise null.
CRDAYSCELSTCPY	INTEGER	If OBJECTTYPE is TS or IX and IMAGECOPY is YES, the number of days since the last image copy. Otherwise null.
CRINDEXSIZE	INTEGER	If OBJECTTYPE is IX and IMAGECOPY is YES, the number of active pages in the index space or partition. Otherwise null.
REORGLASTTIME	TIMESTAMP	Timestamp of the last REORG on the object. Null if REORG was never run, or if the last REORG execution was terminated.
RRTINSDELUPDPCT	INTEGER	If OBJECTTYPE is TS and REORG is YES, the ratio of the sum of insert, update, and delete operations since the last REORG to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.
RRTUNCINSPCT	INTEGER	If OBJECTTYPE is TS and REORG is YES, the ratio of the number of unclustered insert operations to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.
RRTDISORGLOBPCT	INTEGER	If OBJECTTYPE is TS and REORG is YES, the ratio of the number of imperfectly chunked LOBs to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.

Table 163. Result set row for second DSNACCOR result set (continued)

Column name	Data type	Description
RRTMASSDELETE	INTEGER	If OBJECTTYPE is TS, REORG is YES, and the table space is a segmented table space or LOB table space, the number of mass deletes since the last REORG or LOAD REPLACE. If OBJECTTYPE is TS, REORG is YES, and the table space is nonsegmented, the number of dropped tables since the last REORG or LOAD REPLACE. Otherwise null.
RRTINDREF	INTEGER	If OBJECTTYPE is TS, REORG is YES, the ratio of the total number of overflow records that were created since the last REORG or LOAD REPLACE to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.
RRIINSDELPCT	INTEGER	If OBJECTTYPE is IX and REORG is YES, the ratio of the total number of insert and delete operations since the last REORG to the total number of index entries in the index space or partition, expressed as a percentage. Otherwise null.
RRIAPPINSPCT	INTEGER	If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index entries that were inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE that had a key value greater than the maximum key value in the index space or partition, to the number of index entries in the index space or partition, expressed as a percentage. Otherwise null.
RRIPSDDELPCT	INTEGER	If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index entries that were pseudo-deleted (the RID entry was marked as deleted) since the last REORG, REBUILD INDEX, or LOAD REPLACE to the number of index entries in the index space or partition, expressed as a percentage. Otherwise null.
RRIMASSDELETE	INTEGER	If OBJECTTYPE is IX and REORG is YES, the number of mass deletes from the index space or partition since the last REORG, REBUILD, or LOAD REPLACE. Otherwise null.
RRILEAF	INTEGER	If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was far from the location of the original page, to the total number of active pages in the index space or partition, expressed as a percentage. Otherwise null.
RRINUMLEVELS	INTEGER	If OBJECTTYPE is IX and REORG is YES, the number of levels in the index tree that were added or removed since the last REORG, REBUILD INDEX, or LOAD REPLACE. Otherwise null.
STATSLASTTIME	TIMESTAMP	Timestamp of the last RUNSTATS on the object. Null if RUNSTATS was never run, or if the last RUNSTATS execution was terminated.
SRTINSDELUPDPCT	INTEGER	If OBJECTTYPE is TS and RUNSTATS is YES, the ratio of the total number of insert, update, and delete operations since the last RUNSTATS on a table space or partition, to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.
SRTINSDELUPDABS	INTEGER	If OBJECTTYPE is TS and RUNSTATS is YES, the total number of insert, update, and delete operations since the last RUNSTATS on a table space or partition. Otherwise null.

Table 163. Result set row for second DSNACCOR result set (continued)

Column name	Data type	Description
SRTMASSDELETE	INTEGER	If OBJECTTYPE is TS and RUNSTATS is YES, the number of mass deletes from the table space or partition since the last REORG or LOAD REPLACE. Otherwise null.
SRIINSDELPCT	INTEGER	If OBJECTTYPE is IX and RUNSTATS is YES, the ratio of the total number of insert and delete operations since the last RUNSTATS on the index space or partition, to the total number of index entries in the index space or partition, expressed as a percentage. Otherwise null.
SRIINSDELABS	INTEGER	If OBJECTTYPE is IX and RUNSTATS is YES, the number insert and delete operations since the last RUNSTATS on the index space or partition. Otherwise null.
SRIMASSDELETE	INTEGER	If OBJECTTYPE is IX and RUNSTATS is YES, the number of mass deletes from the index space or partition since the last REORG, REBUILD INDEX, or LOAD REPLACE. Otherwise, this value is null.
TOTALEXTENTS	SMALLINT	If EXTENTS is YES, the number of physical extents in the table space, index space, or partition. Otherwise, this value is null.

Appendix C. Advisory or restrictive states

To control access and help ensure data integrity, DB2 sets a restrictive or nonrestrictive (advisory) status on certain objects. This appendix outlines the restrictive and nonrestrictive (advisory) object statuses that affect utilities, and the required steps to correct each status for a particular object.

Use the DISPLAY DATABASE command to display the current status for an object.

The following states are described in this section:

In addition to these states, the output from the DISPLAY DATABASE command might also indicate that an object is in logical page list (LPL) status. This state means that the pages that are listed in the LPL PAGES column are logically in error and are unavailable for access. DB2 writes entries for these pages in an LPL. For more information about an LPL and on how to remove pages from the LPL, see Part 4 of *DB2 Administration Guide*.

Auxiliary CHECK-pending status

The auxiliary CHECK-pending (ACHKP) restrictive status is set on when at least one base table LOB column error is detected and not invalidated as a result of running CHECK DATA AUXERROR REPORT.

Refer to Table 164 for information about resetting the auxiliary CHECK-pending status. This table lists the status name, abbreviation, affected object, and any corrective actions.

Table 164. Resetting auxiliary CHECK-pending status

Status	Abbreviation	Object affected	Corrective action	Notes
Auxiliary CHECK-pending	ACHKP	Base table space	1. Update or delete invalid LOBs using SQL. 2. Run the CHECK DATA utility with the appropriate SCOPE option to verify the validity of LOBs and reset ACHKP status. You can use the REPAIR utility, followed by CHECK DATA, to reset the ACHKP status, but use caution.	1

Notes:

1. A base table space in the ACHKP status is unavailable for processing by SQL.

Auxiliary warning status

Auxiliary warning (AUXW) status is set on when at least one base table LOB column has an invalidated LOB as a result of running CHECK DATA AUXERROR INVALIDATE. An attempt to retrieve an invalidated LOB results in a -904 SQL return code.

The RECOVER utility also sets AUXW status if it finds an invalid LOB column. Invalid LOB columns might result from a situation in which all the following actions occur:
1. LOB table space was defined with LOG NO.
2. LOB table space was recovered.
3. LOB was updated since the last image copy.

Refer to Table 165 for information about resetting the auxiliary warning status. This table lists the status name, abbreviation, affected objects, and any corrective actions.

Table 165. Resetting auxiliary warning status

Status	Abbreviation	Object affected	Corrective action	Notes
Auxiliary warning	AUXW	Base table space	1. Update or delete invalid LOBs using SQL. 2. Run CHECK DATA utility to verify the validity of LOBs and reset AUXW status.	1,2,3
Auxiliary warning	AUXW	LOB table space	1. Update or delete invalid LOBs using SQL. 2. Run CHECK LOB utility to verify the validity of LOBs and reset AUXW status.	1

Notes:

1. A base table space or LOB table space in the AUXW status is available for processing by SQL, even though it contains invalid LOBs. However, an attempt to retrieve an invalid LOB results in a -904 SQL return code.
2. DB2 can access all rows of a base table space that are in the AUXW status. SQL can update the invalid LOB column and delete base table rows, but the value of the LOB column cannot be retrieved. If DB2 attempts to access an invalid LOB column, a -904 SQL code is returned. The AUXW status remains on the base table space even when SQL deletes or updates the last invalid LOB column.
3. If CHECK DATA AUXERROR REPORT encounters only invalid LOB columns and no other LOB column errors, the base table space is set to the auxiliary warning status.

CHECK-pending status

The CHECK-pending (CHKP) restrictive status indicates that an object might be in an inconsistent state and must be checked.

The following utilities set the CHECK-pending status on a table space if referential integrity constraints are encountered:
- LOAD with ENFORCE NO
- RECOVER to a point in time
- CHECK LOB

The CHECK-pending status can also affect a base table space or a LOB table space.

DB2 ignores informational referential integrity constraints and does not set CHECK-pending status for them.

Refer to Table 166 for information about resetting the CHECK-pending status. This table lists the status name, abbreviation, affected objects, and any corrective actions.

Table 166. Resetting CHECK-pending status

Status	Abbreviation	Object affected	Corrective action	Notes
CHECK-pending	CHKP	Table space, base table space	Check and correct referential integrity constraints using the CHECK DATA utility. If a table space is in both REORG-pending and CHECK-pending status (or auxiliary CHECK-pending status), run REORG first and then use CHECK DATA to clear the respective states.	
CHECK-pending	CHKP	Partitioning index, nonpartitioning index, index on the auxiliary table	1. Run CHECK INDEX on the index. 2. If any errors are found, use the REBUILD INDEX utility to rebuild the index from existing data.	1
CHECK-pending	CHKP	LOB table space	Use the CHECK LOB utility to check the LOB table space. If any errors are found: 1. Correct any defects that are found in the LOB table space by using the REPAIR utility. 2. Run CHECK LOB again to reset the CHECK-pending status. 3. See Table 165 on page 854 if an AUXW status exists.	

Notes:

1. An index might be placed in the CHECK-pending status if you recovered an index to a specific RBA or LRSN from a copy and applied the log records, but you did not recover the table space in the same list. The CHECK-pending status can also be placed on an index if you specified the table space and the index in the same list, but the RECOVER point in time was not a QUIESCE or COPY SHRLEVEL REFERENCE point.

COPY-pending status

The COPY-pending (COPY) restrictive status indicates that the affected object must be copied.

DB2 ignores informational referential integrity constraints and does not set CHECK-pending status for them.

Refer to Table 167 for information about resetting the COPY-pending status. This table lists the status name, abbreviation, affected objects, and any corrective actions.

Table 167. Resetting COPY-pending status

Status	Abbreviation	Object affected	Corrective action	Notes
COPY-pending	COPY	Table space, table space partition	Take an image copy of the affected object.	

Group buffer pool RECOVER-pending status

The group buffer pool RECOVER-pending (GRECP) status is set on when a coupling facility fails with pages that were not externalized. The affected object must be recovered.

Refer to Table 168 for information about resetting the group buffer pool RECOVER-pending status. This table lists the status name, abbreviation, affected objects, and any corrective actions.

Table 168. Resetting group buffer pool RECOVER-pending status

Status	Abbreviation	Object affected	Corrective action	Notes
Group buffer pool RECOVER-pending	GRECP	Object	Recover the object, or use START DATABASE to recover the object.	

Informational COPY-pending status

The informational COPY-pending (ICOPY) advisory status indicates that the affected object should be copied.

Refer to Table 169 for information about resetting the informational COPY-pending status. This table lists the status name, abbreviation, affected objects, and any corrective actions.

Table 169. Resetting informational COPY-pending status

Status	Abbreviation	Object affected	Corrective action	Notes
Informational COPY-pending	ICOPY	Partitioning index, nonpartitioning index, index on the auxiliary table	Copy the affected index.	

REBUILD-pending status

A REBUILD-pending restrictive status indicates that the affected index or index partition is broken and must be rebuilt from the data.

REBUILD-pending (RBDP) status indicates that the physical or logical partition is inaccessible and must be rebuilt. RBDP status is set on a data-partitioned secondary index if you create the index after performing the following actions:

- Create a partitioned table space.
- Create a partitioning index.
- Insert a row into a table.

In this situation, the last partition of the table space is set to REORG-pending (REORP) restrictive status.

REBUILD-pending star (RBDP*) status indicates that a logical partition of a nonpartitioned secondary index is unavailable for read-write access and the entire index is unavailable for read access.

Page set REBUILD-pending (PSRBD) status indicates that an entire nonpartitioned secondary index or index on the auxiliary table is unavailable for read-write access.

Rebuilding an index and thereby resetting the REBUILD-pending status invalidates the dynamic statement cache for the related table.

If you alter the data type of a column to a numeric data type, RECOVER INDEX cannot complete. You must rebuild the index.

Refer to Table 170 for information about resetting a REBUILD-pending status. This table lists the status name, abbreviation, affected objects, and any corrective actions.

Table 170. Resetting REBUILD-pending status

Status	Abbreviation	Object affected	Corrective action	Notes
REBUILD-pending	RBDP	Physical or logical index partition	Run the REBUILD utility on the affected index partition.	
REBUILD-pending star	RBDP*	Logical partitions of nonpartitioned secondary indexes	Run REBUILD INDEX PART or RECOVER utility on the affected logical partitions.	
Page set REBUILD-pending	PSRBD	Nonpartitioned secondary index, index on the auxiliary table	Run REBUILD INDEX ALL, the RECOVER utility, or run REBUILD INDEX listing all indexes in the affected index space.	
REBUILD-pending	RBDP, RBDP*, or PSRBD	all	The following actions also reset the REBUILD-pending status: • Use LOAD REPLACE for the table space or partition. • Use REPAIR SET INDEX with NORBDPEND on the index partition. Be aware that this does not correct the data inconsistency in the index partition. Use CHECK INDEX instead of REPAIR to verify referential integrity constraints. • Start the database that contains the index space with ACCESS FORCE. Be aware that this does not correct the data inconsistency in the index partition. • Run REORG INDEX SORTDATA on the affected index.	

RECOVER-pending status

The RECOVER-pending (RECP) restrictive status indicates that a table space or table space partition is broken and must be recovered.

Refer to Table 171 for information about resetting the RECOVER-pending status. This table lists the status name, abbreviation, affected objects, and any corrective actions.

Table 171. Resetting RECOVER-pending status

Status	Abbreviation	Object affected	Corrective action	Notes
RECOVER-pending	RECP	Table space	Run the RECOVER utility on the affected object.	
RECOVER-pending	RECP	Table space partition	Recover the partition.	

Table 171. Resetting RECOVER-pending status (continued)

Status	Abbreviation	Object affected	Corrective action	Notes
RECOVER-pending	RECP	Index on the auxiliary table	Correct the RECOVER-pending status by using one of the following utilities: • REBUILD INDEX • RECOVER INDEX • REORG INDEX SORTDATA	
RECOVER-pending	RECP	Index space	Run one of the following utilities on the affected index space to reset RECP, RBDP, RBDP*, or PSRBDP status: • REBUILD INDEX • RECOVER INDEX • REORG INDEX SORTDATA	
RECOVER-pending	RECP	Any	The following actions also reset the RECOVER-pending status: • Use LOAD REPLACE for the table space or partition. • Use REPAIR SET TABLESPACE or INDEX with NORCVRPEND on the table space or partition. Be aware that this does not correct the data inconsistency in the table space or partition. • Start the database that contains the table space or index space with ACCESS FORCE. Be aware that this does not correct the data inconsistency in the table space or partition.	

REFRESH-pending status

Whenever DB2 marks an object in refresh-pending (REFP) status, it also puts the object in RECOVER-pending (RECP) or REBUILD-pending (RBDP or PSRBD). If a user-defined table space is in refresh-pending (REFP) status, you can replace the data by using LOAD REPLACE. At the successful completion of the RECOVER and LOAD REPLACE jobs, both (REFP and RECP or REFP and RBDP or PSRBD) statuses are reset.

REORG-pending status

The REORG-pending (REORP) restrictive status indicates that a table space partition definition has changed and the affected partitions must be reorganized before the data is accessible.

The REORG-pending (AREO*) advisory status indicates that a table space, index, or partition needs to be reorganized for optimal performance.

Refer to Table 172 on page 859 for information about resetting the REORG-pending status. This table lists the status name, abbreviation, affected objects, and any corrective actions.

Table 172. Resetting REORG-pending status

Status	Abbreviation	Object affected	Corrective action	Notes
REORG-pending	REORP	Table space	Perform one of the following actions: • Use LOAD REPLACE for the entire table space. • Run the REORG TABLESPACE utility with SHRLEVEL NONE. If a table space is in both REORG-pending and CHECK-pending status (or auxiliary CHECK-pending status), run REORG first and then run CHECK DATA to clear the respective states. • Run REORG PART*m*:*n* SHRLEVEL NONE.	
REORG-pending	REORP	Partitioned table space	For row lengths <= 32 KB: 1. Run REORG TABLESPACE SHRLEVEL NONE SORTDATA. For row lengths > 32 KB: 1. Run REORG TABLESPACE UNLOAD ONLY. 2. Run LOAD TABLESPACE FORMAT UNLOAD.	
Advisory REORG-pending	AREO*	Table space	Run one of the following utilities: • REORG TABLESPACE • LOAD REPLACE • REPAIR TABLESPACE	1
Advisory REORG-pending	AREO*	Index space	Run one of the following utilities: • REORG TABLESPACE • LOAD REPLACE • REORG INDEX • REPAIR INDEX	1

Notes:

1. You can reset AREO* for a specific partition without being restricted by another AREO* for an adjacent partition. When you run REPAIR VERSIONS, the utility resets the status and updates the version information in SYSTABLEPART for table spaces and SYSINDEXES for indexes.

Restart-pending status

The restart-pending (RESTP) status is set on if an object has backout work pending at the end of DB2 restart.

Refer to Table 173 on page 860 for information about resetting the restart-pending status. This table lists the status name, abbreviation, affected objects, and any corrective actions.

Resetting an advisory or restrictive status

Table 173. Resetting restart-pending status

Status	Abbreviation	Object affected	Corrective action	Notes
Restart-pending	RESTP	Table space, table space partitions, index spaces, and physical index space partitions	Objects in the RESTP status remain unavailable until backout work is complete, or until restart is canceled and a conditional restart or cold start is performed in its place. See Part 4 (Volume 1) of *DB2 Administration Guide* for information about the RESTP restrictive status.	1,2,3

Notes:

1. Delay running REORG TABLESPACE SHRLEVEL CHANGE until all RESTP statuses are reset.
2. You cannot use LOAD REPLACE on an object that is in the RESTP status.
3. Utility activity against page sets or partitions with RESTP status is not allowed. Any attempt to access a page set or partition with RESTP status terminates with return code 8.

Appendix D. Running the productivity-aid sample programs

DB2 provides four sample programs that many users find helpful as productivity aids. These programs are shipped as source code, so you can modify them to meet your needs. The programs are:

DSNTIAUL The sample unload program. This program, which is written in assembler language, is a simple alternative to the UNLOAD utility. It unloads some or all rows from up to 100 DB2 tables. With DSNTIAUL, you can unload data of any DB2 built-in data type or distinct type. You can unload up to 32 KB of data from a LOB column. DSNTIAUL unloads the rows in a form that is compatible with the LOAD utility and generates utility control statements for LOAD. DSNTIAUL also lets you execute any SQL non-SELECT statement that can be executed dynamically. See "Running DSNTIAUL" on page 862.

DSNTIAD A sample dynamic SQL program that is written in assembler language. With this program, you can execute any SQL statement that can be executed dynamically, except a SELECT statement. See "Running DSNTIAD" on page 866.

DSNTEP2 A sample dynamic SQL program that is written in the PL/I language. With this program, you can execute any SQL statement that can be executed dynamically. You can use the source version of DSNTEP2 and modify it to meet your needs, or, if you do not have a PL/I compiler at your installation, you can use the object code version of DSNTEP2. See "Running DSNTEP2 and DSNTEP4" on page 868.

| **DSNTEP4** A sample dynamic SQL program that is written in the PL/I language. This program is identical to DSNTEP2 except DSNTEP4 uses multi-row fetch for increased performance. You can use the source version of DSNTEP4 and modify it to meet your needs, or, if you do not have a PL/I compiler at your installation, you can use the object code version of DSNTEP4. See "Running DSNTEP2 and DSNTEP4" on page 868.

Because these four programs also accept the static SQL statements CONNECT, SET CONNECTION, and RELEASE, you can use the programs to access DB2 tables at remote locations.

Retrieval of UTF-16 Unicode data: You can use DSNTEP2, DSNTEP4, and
DSNTIAUL to retrieve Unicode UTF-16 graphic data. However, these programs
might not be able to display some characters, if those characters have no mapping
in the target SBCS EBCDIC CCSID.

DSNTIAUL and DSNTIAD are shipped only as source code, so you must precompile, assemble, link, and bind them before you can use them. If you want to use the source code version of DSNTEP2 or DSNTEP4, you must precompile, compile, link, and bind it. You need to bind the object code version of DSNTEP2 or DSNTEP4 before you can use it. Usually a system administrator prepares the programs as part of the installation process. Table 174 on page 862 indicates which installation job prepares each sample program. All installation jobs are in data set DSN810.SDSNSAMP.

Table 174. Jobs that prepare DSNTIAUL, DSNTIAD, DSNTEP2, and DSNTEP4

Program name	Program preparation job
DSNTIAUL	DSNTEJ2A
DSNTIAD	DSNTIJTM
DSNTEP2 (source)	DSNTEJ1P
DSNTEP2 (object)	DSNTEJ1L
DSNTEP4 (source)	DSNTEJ1P
DSNTEP4 (object)	DSNTEJ1L

To run the sample programs, use the DSN RUN command, which is described in detail in Chapter 2 of *DB2 Command Reference*. Table 175 lists the load module name and plan name that you must specify, and the parameters that you can specify when you run each program. See the following sections for the meaning of each parameter.

Table 175. DSN RUN option values for DSNTIAUL, DSNTIAD, DSNTEP2, and DSNTEP4

Program name	Load module	Plan	Parameters
DSNTIAUL	DSNTIAUL	DSNTIB81	SQL *number of rows per fetch* # TOLWARN(NO | YES)
DSNTIAD	DSNTIAD	DSNTIA81	RC0 SQLTERM(*termchar*)
DSNTEP2	DSNTEP2	DSNTEP81	ALIGN(MID) or ALIGN(LHS) NOMIXED or MIXED SQLTERM(*termchar*) # TOLWARN(NO | YES)
DSNTEP4	DSNTEP4	DSNTP481	ALIGN(MID) or ALIGN(LHS) NOMIXED or MIXED SQLTERM(*termchar*) # TOLWARN(NO | YES)

The remainder of this chapter contains the following information about running each program:

- Descriptions of the input parameters
- Data sets that you must allocate before you run the program
- Return codes from the program
- Examples of invocation

See the sample jobs that are listed in Table 174 for a working example of each program.

Running DSNTIAUL

This section contains information that you need when you run DSNTIAUL, including parameters, data sets, return codes, and invocation examples.

To retrieve data from a remote site by using the multi-row fetch capability for
enhanced performance, bind DSNTIAUL with the DBPROTOCOL(DRDA) option.

To run DSNTIAUL remotely when it is bound with the DBPROTOCOL(PRIVATE) option, switch DSNTIAUL to single-row fetch mode by specifying 1 for the number of rows per fetch parameter.

DSNTIAUL parameters:

SQL
Specify SQL to indicate that your input data set contains one or more complete SQL statements, each of which ends with a semicolon. You can include any SQL statement that can be executed dynamically in your input data set. In addition, you can include the static SQL statements CONNECT, SET CONNECTION, or RELEASE. DSNTIAUL uses the SELECT statements to determine which tables to unload and dynamically executes all other statements except CONNECT, SET CONNECTION, and RELEASE. DSNTIAUL executes CONNECT, SET CONNECTION, and RELEASE statically to connect to remote locations.

number of rows per fetch
Specify a number from 1 to 32767 to specify the number of rows per fetch that DSNTIAUL retrieves. If you do not specify this number, DSNTIAUL retrieves 100 rows per fetch. This parameter can be specified with the SQL parameter.

Specify 1 to retrieve data from a remote site when DSNTIAUL is bound with the DBPROTOCOL(PRIVATE) option.

TOLWARN
Specify NO (the default) or YES to indicate whether DSNTIAUL continues to retrieve rows after receiving an SQL warning:

NO If a warning occurs when DSNTIAUL executes an OPEN or FETCH to retrieve rows, DSNTIAUL stops retrieving rows. If the SQLWARN1, SQLWARN2, SQLWARN6, or SQLWARN7 flag is set when DSNTIAUL executes a FETCH to retrieve rows, DSNTIAUL continues to retrieve rows.

Exception:

YES If a warning occurs when DSNTIAUL executes an OPEN or FETCH to retrieve rows, DSNTIAUL continues to retrieve rows.

If you do not specify the SQL parameter, your input data set must contain one or more single-line statements (without a semicolon) that use the following syntax:

```
table or view name [WHERE conditions] [ORDER BY columns]
```

Each input statement must be a valid SQL SELECT statement with the clause SELECT * FROM omitted and with no ending semicolon. DSNTIAUL generates a SELECT statement for each input statement by appending your input line to SELECT * FROM, then uses the result to determine which tables to unload. For this input format, the text for each table specification can be a maximum of 72 bytes and must not span multiple lines.

You can use the input statements to specify SELECT statements that join two or more tables or select specific columns from a table. If you specify columns, you need to modify the LOAD statement that DSNTIAUL generates.

DSNTIAUL data sets:

Data set	Description
SYSIN	Input data set.

You cannot enter comments in DSNTIAUL input.

The record length for the input data set must be at least 72 bytes. DSNTIAUL reads only the first 72 bytes of each record.

SYSPRINT Output data set. DSNTIAUL writes informational and error messages in this data set.

The record length for the SYSPRINT data set is 121 bytes.

SYSPUNCH Output data set. DSNTIAUL writes the LOAD utility control statements in this data set.

SYSREC*nn* Output data sets. The value *nn* ranges from 00 to 99. You can have a maximum of 100 output data sets for a single execution of DSNTIAUL. Each data set contains the data that is unloaded when DSNTIAUL processes a SELECT statement from the input data set. Therefore, the number of output data sets must match the number of SELECT statements (if you specify parameter SQL) or table specifications in your input data set.

Define all data sets as sequential data sets. You can specify the record length and block size of the SYSPUNCH and SYSREC*nn* data sets. The maximum record length for the SYSPUNCH and SYSREC*nn* data sets is 32760 bytes.

DSNTIAUL return codes:

Table 176. DSNTIAUL return codes

Return code	Meaning
0	Successful completion.
4	An SQL statement received a warning code. If the SQL statement was a SELECT statement, DB2 did not perform the associated unload operation. If DB2 return a +394, which indicates that it is using optimization hints, DB2 performs the unload operation.
8	An SQL statement received an error code. If the SQL statement was a SELECT statement, DB2 did not perform the associated unload operation.
12	DSNTIAUL could not open a data set, an SQL statement returned a severe error code (-8*nn* or -9*nn*), or an error occurred in the SQL message formatting routine.

Examples of DSNTIAUL invocation: Suppose that you want to unload the rows for department D01 from the project table. Because you can fit the table specification on one line, and you do not want to execute any non-SELECT statements, you do not need the SQL parameter. Your invocation looks like the one that is shown in Figure 162 on page 865:

```
//UNLOAD  EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIB81) -
       LIB('DSN810.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DSN=DSN8UNLD.SYSREC00,
//            UNIT=SYSDA,SPACE=(32760,(1000,500)),DISP=(,CATLG),
//            VOL=SER=SCR03
//SYSPUNCH DD DSN=DSN8UNLD.SYSPUNCH,
//            UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
//            VOL=SER=SCR03,RECFM=FB,LRECL=120,BLKSIZE=1200
//SYSIN    DD *
DSN8810.PROJ WHERE DEPTNO='D01'
```

Figure 162. DSNTIAUL invocation without the SQL parameter

If you want to obtain the LOAD utility control statements for loading rows into a table, but you do not want to unload the rows, you can set the data set names for the SYSREC*nn* data sets to DUMMY. For example, to obtain the utility control statements for loading rows into the department table, you invoke DSNTIAUL as shown in Figure 163:

```
//UNLOAD  EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIB81) -
       LIB('DSN810.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DUMMY
//SYSPUNCH DD DSN=DSN8UNLD.SYSPUNCH,
//            UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
//            VOL=SER=SCR03,RECFM=FB,LRECL=120,BLKSIZE=1200
//SYSIN    DD *
DSN8810.DEPT
```

Figure 163. DSNTIAUL invocation to obtain LOAD control statements

Now suppose that you also want to use DSNTIAUL to do these things:

- Unload all rows from the project table
- Unload only rows from the employee table for employees in departments with department numbers that begin with D, and order the unloaded rows by employee number
- Lock both tables in share mode before you unload them
- Retrieve 250 rows per fetch

For these activities, you must specify the SQL parameter and specify the number of rows per fetch when you run DSNTIAUL. Your DSNTIAUL invocation is shown in Figure 164 on page 866:

```
//UNLOAD  EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIB81) PARMS('SQL,250') -
       LIB('DSN810.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DSN=DSN8UNLD.SYSREC00,
//            UNIT=SYSDA,SPACE=(32760,(1000,500)),DISP=(,CATLG),
//            VOL=SER=SCR03
//SYSREC01 DD DSN=DSN8UNLD.SYSREC01,
//            UNIT=SYSDA,SPACE=(32760,(1000,500)),DISP=(,CATLG),
//            VOL=SER=SCR03
//SYSPUNCH DD DSN=DSN8UNLD.SYSPUNCH,
//            UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
//            VOL=SER=SCR03,RECFM=FB,LRECL=120,BLKSIZE=1200
//SYSIN    DD *
LOCK TABLE DSN8810.EMP IN SHARE MODE;
LOCK TABLE DSN8810.PROJ IN SHARE MODE;
SELECT * FROM DSN8810.PROJ;
SELECT * FROM DSN8810.EMP
  WHERE WORKDEPT LIKE 'D%'
  ORDER BY EMPNO;
```

Figure 164. DSNTIAUL invocation with the SQL parameter

Running DSNTIAD

This section contains information that you need when you run DSNTIAD, including parameters, data sets, return codes, and invocation examples.

DSNTIAD parameters:

RC0
If you specify this parameter, DSNTIAD ends with return code 0, even if the program encounters SQL errors. If you do not specify RC0, DSNTIAD ends with a return code that reflects the severity of the errors that occur. Without RC0, DSNTIAD terminates if more than 10 SQL errors occur during a single execution.

SQLTERM(*termchar***)**
Specify this parameter to indicate the character that you use to end each SQL statement. You can use any special character **except** one of those listed in Table 177. SQLTERM(;) is the default.

Table 177. Invalid special characters for the SQL terminator

Name	Character	Hexadecimal representation
blank		X'40'
comma	,	X'6B'
double quotation mark	"	X'7F'
left parenthesis	(	X'4D'
right parenthesis	)	X'5D'
single quotation mark	'	X'7D'
underscore	_	X'6D'

Use a character other than a semicolon if you plan to execute a statement that contains embedded semicolons.

Example: Suppose that you specify the parameter SQLTERM(#) to indicate that the character # is the statement terminator. Then a CREATE TRIGGER statement with embedded semicolons looks like this:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END#
```

A CREATE PROCEDURE statement with embedded semicolons looks like the following statement:

```
CREATE PROCEDURE PROC1 (IN PARM1 INT, OUT SCODE INT)
  LANGUAGE SQL
  BEGIN
    DECLARE SQLCODE INT;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
      SET SCODE = SQLCODE;
    UPDATE TBL1 SET COL1 = PARM1;
  END #
```

Be careful to choose a character for the statement terminator that is not used within the statement.

DSNTIAD data sets:

Data set	Description
SYSIN	Input data set. In this data set, you can enter any number of non-SELECT SQL statements, each terminated with a semicolon. A statement can span multiple lines, but DSNTIAD reads only the first 72 bytes of each line. You cannot enter comments in DSNTIAD input.
SYSPRINT	Output data set. DSNTIAD writes informational and error messages in this data set. DSNTIAD sets the record length of this data set to 121 bytes and the block size to 1210 bytes.

Define all data sets as sequential data sets.

DSNTIAD return codes:

Table 178. DSNTIAD return codes

Return code	Meaning
0	Successful completion, or the user-specified parameter RC0.
4	An SQL statement received a warning code.
8	An SQL statement received an error code.
12	DSNTIAD could not open a data set, the length of an SQL statement was more than 32760 bytes, an SQL statement returned a severe error code (-8*nn* or -9*nn*), or an error occurred in the SQL message formatting routine.

Example of DSNTIAD invocation: Suppose that you want to execute 20 UPDATE statements, and you do not want DSNTIAD to terminate if more than 10 errors

occur. Your invocation looks like the one that is shown in Figure 165:

```
//RUNTIAD EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA81) PARMS('RC0') -
       LIB('DSN810.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
UPDATE DSN8810.PROJ SET DEPTNO='J01' WHERE DEPTNO='A01';
UPDATE DSN8810.PROJ SET DEPTNO='J02' WHERE DEPTNO='A02';
.
.
.
UPDATE DSN8810.PROJ SET DEPTNO='J20' WHERE DEPTNO='A20';
```

Figure 165. DSNTIAD Invocation with the RC0 Parameter

Running DSNTEP2 and DSNTEP4

This section contains information that you need when you run DSNTEP2 or DSNTEP4, including parameters, data sets, return codes, and invocation examples.

DSNTEP2 and DSNTEP4 write their results to the data set that is defined by the SYSPRINT DD statement. SYSPRINT data must have a logical record length of 133 bytes (LRECL=133). Otherwise, the program issues return code 12 with abend U4038 and reason code 1. This abend occurs due to the PL/I file exception error IBM0201S ONCODE=81. The following error message is issued: The UNDEFINEDFILE condition was raised because of conflicting DECLARE and OPEN attributes (FILE= SYSPRINT).

Note: When you allocate a new data set with the SYSPRINT DD statement, either specify a DCB with LRECL=133, or do not specify the DCB parameter.

DSNTEP2 and DSNTEP4 parameters:

ALIGN(MID) or ALIGN(LHS)
Specifies the alignment.

ALIGN(MID)
Specifies that DSNTEP2 or DSNTEP4 output should be centered. **ALIGN(MID)** is the default.

ALIGN(LHS)
Specifies that the DSNTEP2 or DSNTEP4 output should be left-justified.

NOMIXED or MIXED
Specifies whether DSNTEP2 or DSNTEP4 contains any DBCS characters.

NOMIXED
Specifies that the DSNTEP2 or DSNTEP4 input contains no DBCS characters. **NOMIXED** is the default.

MIXED
Specifies that the DSNTEP2 or DSNTEP4 input contains some DBCS characters.

SQLTERM(*termchar***)**
Specifies the character that you use to end each SQL statement. You can use any character **except** one of those that are listed in Table 177 on page 866. **SQLTERM(;)** is the default.

Use a character other than a semicolon if you plan to execute a statement that contains embedded semicolons.

Example: Suppose that you specify the parameter SQLTERM(#) to indicate that the character # is the statement terminator. Then a CREATE TRIGGER statement with embedded semicolons looks like this:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END#
```

A CREATE PROCEDURE statement with embedded semicolons looks like the following statement:

```
CREATE PROCEDURE PROC1 (IN PARM1 INT, OUT SCODE INT)
  LANGUAGE SQL
  BEGIN
    DECLARE SQLCODE INT;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
      SET SCODE = SQLCODE;
    UPDATE TBL1 SET COL1 = PARM1;
  END #
```

Be careful to choose a character for the statement terminator that is not used within the statement.

If you want to change the SQL terminator within a series of SQL statements, you can use the --#SET TERMINATOR control statement.

Example: Suppose that you have an existing set of SQL statements to which you want to add a CREATE TRIGGER statement that has embedded semicolons. You can use the default SQLTERM value, which is a semicolon, for all of the existing SQL statements. Before you execute the CREATE TRIGGER statement, include the --#SET TERMINATOR # control statement to change the SQL terminator to the character #:

```
SELECT * FROM DEPT;
SELECT * FROM ACT;
SELECT * FROM EMPPROJACT;
SELECT * FROM PROJ;
SELECT * FROM PROJACT;
--#SET TERMINATOR #
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END#
```

See the following discussion of the SYSIN data set for more information about the --#SET control statement.

TOLWARN

Specify NO (the default) or YES to indicate whether DSNTEP2 or DSNTEP4 continues to process SQL SELECT statements after receiving an SQL warning:

NO If a warning occurs when DSNTEP2 or DSNTEP4 executes an OPEN or FETCH for a SELECT statement, DSNTEP2 or DSNTEP4 stops processing the SELECT statement. If SQLCODE +445 or SQLCODE +595 occurs when DSNTEP2 or DSNTEP4 executes a FETCH for a SELECT statement, DSNTEP2 or DSNTEP4 continues to process the SELECT statement. If SQLCODE +802 occurs when DSNTEP2 or DSNTEP4 executes a FETCH for a SELECT statement, DSNTEP2 or

DSNTEP4 continues to process the SELECT statement if the TOLARTHWRN control statement is set to YES.

YES If a warning occurs when DSNTEP2 or DSNTEP4 executes an OPEN or FETCH for a SELECT statement, DSNTEP2 or DSNTEP4 continues to process the SELECT statement.

DSNTEP2 and DSNTEP4 data sets:

Data Set	Description
SYSIN	Input data set. In this data set, you can enter any number of SQL statements, each terminated with a semicolon. A statement can span multiple lines, but DSNTEP2 or DSNTEP4 reads only the first 72 bytes of each line. You can enter comments in DSNTEP2 or DSNTEP4 input with an asterisk (*) in column 1 or two hyphens (--) anywhere on a line. Text that follows the asterisk is considered to be comment text. Text that follows two hyphens can be comment text or a control statement. Comments are not considered in dynamic statement caching. Comments and control statements cannot span lines. You can enter control statements of the following form in the DSNTEP2 and DSNTEP4 input data set: `--#SET control-option value` The control options are: **TERMINATOR** The SQL statement terminator. *value* is any single-byte character other than one of those that are listed in Table 177 on page 866. The default is the value of the SQLTERM parameter. **ROWS_FETCH** The number of rows that are to be fetched from the result table. *value* is a numeric literal between -1 and the number of rows in the result table. -1 means that all rows are to be fetched. The default is -1. **ROWS_OUT** The number of fetched rows that are to be sent to the output data set. *value* is a numeric literal between -1 and the number of fetched rows. -1 means that all fetched rows are to be sent to the output data set. The default is -1. **MULT_FETCH** This option is valid only for DSNTEP4. Use MULT_FETCH to specify the number of rows that are to be fetched at one time from the result table. The default fetch amount for DSNTEP4 is 100 rows, but you can specify from 1 to 32676 rows. **TOLWARN** Indicates whether DSNTEP2 and DSNTEP4 continue to process an SQL SELECT after an SQL warning is returned. *value* is either NO (the default) or YES. **TOLARTHWRN** Indicates whether DSNTEP2 and DSNTEP4 continue to process

an SQL SELECT statement after an arithmetic SQL warning (SQLCODE +802) is returned. *value* is either NO (the default) or YES.

MAXERRORS
Specifies that number of errors that DSNTEP2 and DSNTEP4 handle before processing stops. The default is 10.

SYSPRINT Output data set. DSNTEP2 and DSNTEP4 write informational and error messages in this data set. DSNTEP2 and DSNTEP4 write output records of no more than 133 bytes.

Define all data sets as sequential data sets.

DSNTEP2 and DSNTEP4 return codes:

Table 179. DSNTEP2 and DSNTEP4 return codes

Return code	Meaning
0	Successful completion.
4	An SQL statement received a warning code.
8	An SQL statement received an error code.
12	The length of an SQL statement was more than 32760 bytes, an SQL statement returned a severe error code (-8*nn* or -9*nn*), or an error occurred in the SQL message formatting routine.

Example of DSNTEP2 invocation: Suppose that you want to use DSNTEP2 to execute SQL SELECT statements that might contain DBCS characters. You also want left-aligned output. Your invocation looks like the one in Figure 166:

```
//RUNTEP2 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTEP2) PLAN(DSNTEP81) PARMS('/ALIGN(LHS) MIXED') -
       LIB('DSN810.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
SELECT * FROM DSN8810.PROJ;
```

Figure 166. DSNTEP2 invocation with the ALIGN(LHS) and MIXED parameters

Example of DSNTEP4 invocation: Suppose that you want to use DSNTEP4 to execute SQL SELECT statements that might contain DBCS characters, and you want center-aligned output. You also want DSNTEP4 to fetch 250 rows at a time. Your invocation looks like the one in Figure 167 on page 872:

```
//RUNTEP2 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTEP4) PLAN(DSNTP481) PARMS('/ALIGN(MID) MIXED') -
       LIB('DSN810.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
--#SET MULT_FETCH 250
SELECT * FROM DSN8810.EMP;
```

Figure 167. DSNTEP4 invocation with the ALIGN(MID) and MIXED parameters and using the MULT_FETCH control option

Appendix E. Real-time statistics tables

The information under this heading is Product-sensitive Programming Interface and Associated Guidance Information, as defined in "Notices" on page 905.

DB2 collects statistics that you can use to determine when you need to perform certain maintenance functions on your table spaces and index spaces.

DB2 collects the statistics in real time. You create tables into which DB2 periodically writes the statistics. You can then write applications that query the statistics and help you decide when to run REORG, RUNSTATS, or COPY, or to enlarge your data sets. Figure 168 shows an overview of the process of collecting and using real-time statistics.

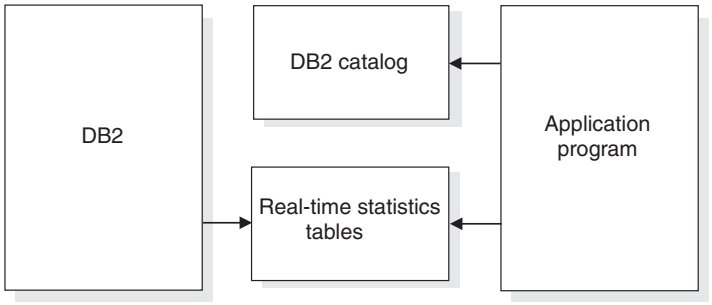


Figure 168. Real-time statistics overview

The following sections provide detailed information about the real-time statistics tables:

- "Setting up your system for real-time statistics"
- "Contents of the real-time statistics tables" on page 875
- "Operating with real-time statistics" on page 887

For information about a DB2-supplied stored procedure that queries the real-time statistics tables, see "The DB2 real-time statistics stored procedure" on page 830.

Setting up your system for real-time statistics

DB2 always generates in-memory statistics for each table space, index space, including catalog objects but not the directory, in your system. For partitioned spaces, DB2 generates information for each partition. However, you need to perform the following steps before DB2 externalizes the statistics to DB2 tables:

1. Create the real-time statistics objects. See "Creating and altering the real-time statistics objects" on page 874.
2. Set the interval for writing statistics. See "Setting the interval for writing real-time statistics" on page 875.
3. Start the real-time statistics database. See "Starting the real-time statistics database" on page 875.
4. Establish base values for the statistics. See "Establishing base values for real-time statistics" on page 875.

Creating and altering the real-time statistics objects

You need to create a database, table space, tables, and indexes for the real-time statistics. Those objects are listed in Table 180. Use the SQL statements in member DSNTESS of data set DSN810.SDSNSAMP as a model for creating the real-time statistics objects. You can create these objects in user-managed or DB2-managed data sets.

Restrictions on changing the provided definitions for the real-time statistics objects: You can change most of the attributes in the provided definitions of the real-time statistics objects. However, you cannot change the following items:

- Object names. You must use the names that are specified in DSNTESS for the database, table space, tables, indexes, and table columns.
- The CCSID parameter on the CREATE DATABASE, CREATE TABLESPACE, and CREATE TABLE statements. The CCSID must be EBCDIC.
- Number of columns or column definitions. You cannot add table columns or modify column definitions.
- The LOCKSIZE parameter on the CREATE TABLESPACE statement. The LOCKSIZE must be ROW.

Note: If Real Time Statistics is active and the tablespaces are created out of order, several utilities might issue abend04E with reason code 00C90101. When you create tablespaces, always create tablespacestats before creating indexspacestats. See APAR PQ73249 for additional information.

Before you can alter an object in the real-time statistics database, you must stop the database. Otherwise, you receive an SQL error. Table 180 shows the DB2 objects for storing real-time statistics.

Table 180. DB2 objects for storing real-time statistics

Object name	Description
DSNRTSDB	Database for real-time statistics objects
DSNRTSTS	Table space for real-time statistics objects
SYSIBM.TABLESPACESTATS	Table for statistics on table spaces and table space partitions
SYSIBM.INDEXSPACESTATS	Table for statistics on index spaces and index space partitions
SYSIBM.TABLESPACESTATS_IX	Unique index on SYSIBM.TABLESPACESTATS (columns DBID, PSID, and PARTITION)
SYSIBM.INDEXSPACESTATS_IX	Unique index on SYSIBM.INDEXSPACESTATS (columns DBID, ISOBID, and PARTITION)

To create the real-time statistics objects, you need the authority to create tables and indexes on behalf of the SYSIBM authorization ID.

DB2 inserts one row in the table for each partition or non-partitioned table space or index space. You therefore need to calculate the amount of disk space that you need for the real-time statistics tables based on the current number of table spaces and indexes in your subsystem.

To determine the amount of storage that you need for the real-time statistics when they are in memory, use the following formula:

```
Max_concurrent_objects_updated * 152 bytes = Storage_in_bytes
```

Where Max_concurrent_objects_updated is the peak number of objects that might be updated concurrently and 152 bytes is the amount of in-memory space that DB2 uses for each object.

Recommendation: Place the statistics indexes and tables in their own buffer pool. When the statistics pages are in memory, the speed at which in-memory statistics are written to the tables improves.

Setting the interval for writing real-time statistics

You can set the interval for writing real-time statistics when you install DB2, and you can subsequently update that interval online. The installation field is REAL TIME STATS on panel DSNTIPO. The default interval is 30 minutes. To update the interval, modify system parameter STATSINT.

In a data sharing environment, each member has its own interval for writing real-time statistics.

Starting the real-time statistics database

After you create the real-time statistics database, DB2 puts it into a stopped state. After you create all the objects in the database, you need to issue START DATABASE(DSNRTSDB) to explicitly start the database.

You must start the database in read-write modeso that DB2 can externalize real-time statistics. See "When DB2 externalizes real-time statistics" on page 888 for information about the conditions for which DB2 externalizes the statistics.

Establishing base values for real-time statistics

Many columns in the real-time statistics tables show the number of times an operation was performed between the last time a particular utility was run and when the real-time statistics are written. For example, STATSINSERT in TABLESPACESTATS indicates the number of records or LOBs that have been inserted after the last RUNSTATS utility was run on the table space or partition. Therefore, for each object for which you want real-time statistics, run the appropriate utility (REORG, RUNSTATS, LOAD REPLACE, REBUILD INDEX, or COPY) to establish a base value from which the delta value can be calculated.

Contents of the real-time statistics tables

The SYSIBM.TABLESPACESTATS table contains statistics information about table spaces and table space partitions. The SYSIBM.INDEXSPACESTATS table contains statistics information about index spaces and index space partitions.

Table 181 describes the columns of the TABLESPACESTATS table and explains how you can use them in deciding when to run REORG, RUNSTATS, or COPY.

Table 181. Descriptions of columns in the TABLESPACESTATS table

Column name	Data type	Description
DBNAME	CHAR(8) NOT NULL	The name of the database. This column is used to map a database to its statistics.
NAME	CHAR(8) NOT NULL	The name of the table space. This column is used to map a table space to its statistics.

Table 181. Descriptions of columns in the TABLESPACESTATS table (continued)

Column name	Data type	Description
PARTITION	SMALLINT NOT NULL	The data set number within the table space. This column is used to map a data set number in a table space to its statistics. For partitioned table spaces, this value corresponds to the partition number for a single partition. For nonpartitioned table spaces, this value is 0.
DBID	SMALLINT NOT NULL	The internal identifier of the database. This column is used to map a DBID to its statistics.
PSID	SMALLINT NOT NULL	The internal identifier of the table space page set descriptor. This column is used to map a PSID to its statistics.
UPDATESTATSTIME	TIMESTAMP NOT NULL WITH DEFAULT	The timestamp when the row was inserted or last updated. This column is updated with the current timestamp when a row in the TABLESPACESTATS table is inserted or updated. You can use this column in several ways: • To determine the actions that caused the latest change to the table. Do this by selecting any of the timestamp columns and comparing them to the UPDATESTATSTIME column. • To determine whether an analysis of data is needed. This determination might be based on a given time interval, or on a combination of the time interval and the amount of activity. For example, suppose that you want to analyze statistics for the last seven days. To determine whether there has been any activity in the past seven days, check whether the difference between the current date and the UPDATESTATSTIME value is less than or equal to seven: `(JULIAN_DAY(CURRENT DATE)-JULIAN_DAY(UPDATESTATSTIME))<= 7`
TOTALROWS	FLOAT	The number of rows or LOBs in the table space or partition. If the table space contains more than one table, this value is the sum of all rows in all tables. A null value means that the number of rows is unknown or that REORG or LOAD has never been run. Use the TOTALROWS value with the value of any column that contains some affected rows to determine the percentage of rows that are affected by a particular action.
NACTIVE	INTEGER	The number of active pages in the table space or partition. A null value means that the number of active pages is unknown. This value is equivalent to the number of preformatted pages. For multi-piece table spaces, this value is the total number of preformatted pages in all data sets. Use the NACTIVE value with the value of any column that contains some affected pages to determine the percentage of pages that are affected by a particular action. For example, suppose that your site's maintenance policies require that COPY is to be run after 20% of the pages in a table space have changed. To determine if a COPY might be required, calculate the ratio of updated pages since the last COPY to the total number of active pages. If the percentage is greater than 20, you need to run COPY: `((COPYUPDATEDPAGES*100)/NACTIVE)>20`

Table 181. Descriptions of columns in the TABLESPACESTATS table (continued)

Column name	Data type	Description
SPACE	INTEGER	The amount of space, in KB, that is allocated to the table space or partition. For multi-piece linear page sets, this value is the amount of space in all data sets. A null value means the amount of space is unknown. Use this value to monitor growth and validate design assumptions.
EXTENTS	SMALLINT	The number of extents in the table space or partition. For multi-piece table spaces, this value is the number of extents for the last data set. For a data set that is striped across multiple volumes, the value is the number of logical extents. A null value means that the number of extents is unknown. Use this value to determine: • When the primary or secondary allocation value for a table space or partition needs to be altered. • When you are approaching the maximum number of extents and risking extend failures.
LOADRLASTTIME	TIMESTAMP	The timestamp of the last LOAD REPLACE on the table space or partition. A null value means that LOAD REPLACE has never been run on the table space or partition or that the timestamp of the last LOAD REPLACE is unknown. You can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last LOAD REPLACE is more recent than the last COPY, you might need to run COPY: `(JULIAN_DAY(LOADRLASTTIME)>JULIAN_DAY(COPYLASTTIME))`
REORGLASTTIME	TIMESTAMP	The timestamp of the last REORG on the table space or partition. A null value means REORG has never been run on the table space or partition or that the timestamp of the last REORG is unknown. You can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last REORG is more recent than the last COPY, you might need to run COPY: `(JULIAN_DAY(REORGLASTTIME)>JULIAN_DAY(COPYLASTTIME))`
REORGINSERTS	INTEGER	The number of records or LOBs that have been inserted since the last REORG or LOAD REPLACE on the table space or partition. A null value means that the number of inserted records or LOBs is unknown.
REORGDELETES	INTEGER	The number of records or LOBs that have been deleted since the last REORG or LOAD REPLACE on the table space or partition. A null value means that the number of deleted records or LOBs is unknown.

Table 181. Descriptions of columns in the TABLESPACESTATS table (continued)

Column name	Data type	Description
REORGUPDATES	INTEGER	The number of rows that have been updated since the last REORG or LOAD REPLACE on the table space or partition. This value does not include LOB updates because LOB updates are really deletions followed by insertions. A null value means that the number of updated rows is unknown. This value can be used with REORGDELETES and REORGINSERTS to determine if a REORG is necessary. For example, suppose that your site's maintenance policies require that REORG is run after 20 per cent of the rows in a table space have changed. To determine if a REORG is required, calculate the sum of updated, inserted, and deleted rows since the last REORG. Then calculate the ratio of that sum to the total number of rows. If the percentage is greater than 20, you might need to run REORG: `(((REORGINSERTS+REORGDELETES+REORGUPDATES)*100)/TOTALROWS)>20`
REORGDISORGLOB	INTEGER	The number of LOBs that were inserted since the last REORG or LOAD REPLACE that are not perfectly chunked. A LOB is perfectly chunked if the allocated pages are in the minimum number of chunks. A null value means that the number of imperfectly chunked LOBs is unknown. Use this value to determine whether you need to run REORG. For example, you might want to run REORG if the ratio of REORGDISORGLOB to the total number of LOBs is greater than 10%: `((REORGDISORGLOB*100)/TOTALROWS)>10`
REORGUNCLUSTINS	INTEGER	The number of records that were inserted since the last REORG or LOAD REPLACE that are not well-clustered with respect to the clustering index. A record is well-clustered if the record is inserted into a page that is within 16 pages of the ideal candidate page. The clustering index determines the ideal candidate page. A null value means that the number of badly-clustered pages is unknown. You can use this value to determine whether you need to run REORG. For example, you might want to run REORG if the following comparison is true: `((REORGUNCLUSTINS*100)/TOTALROWS)>10`
REORGMASSDELETE	INTEGER	The number of mass deletes from a segmented or LOB table space, or the number of dropped tables from a segmented table space, since the last REORG or LOAD REPLACE. A null value means that the number of mass deletes is unknown. If this value is non-zero, a REORG might be necessary.
REORGNEARINDREF	INTEGER	The number of overflow records that were created since the last REORG or LOAD REPLACE and were relocated near the pointer record. For nonsegmented table spaces, a page is near the present page if the two page numbers differ by 16 or less. For segmented table spaces, a page is near the present page if the two page numbers differ by SEGSIZE*2 or less. A null value means that the number of overflow records near the pointer record is unknown.

Table 181. Descriptions of columns in the TABLESPACESTATS table (continued)

Column name	Data type	Description

REORGFARINDEF	INTEGER	The number of overflow records that were created since the last REORG or LOAD REPLACE and were relocated far from the pointer record. For nonsegmented table spaces, a page is far from the present page if the two page numbers differ by more than 16. For segmented table spaces, a page is far from the present page if the two page numbers differ by at least (SEGSIZE*2)+1.
		
		A null value means that the number of overflow records far from the pointer record is unknown.
		
		For example, in a non-data sharing environment, you might run REORG if the following comparison is true:
		`(((REORGNEARINDREF+REORGFARINDREF)*100)/TOTALROWS)>10`
		
		In a data sharing environment, you might run REORG if the following comparison is true:
		`(((REORGNEARINDREF+REORGFARINDREF)*100)/TOTALROWS)>5`

STATSLASTTIME	TIMESTAMP	The timestamp of the last RUNSTATS on the table space or partition. A null value means that RUNSTATS has never been run on the table space or partition, or that the timestamp of the last RUNSTATS is unknown. You can compare this timestamp to the timestamp of the last REORG on the same object to determine when RUNSTATS is needed. If the date of the last REORG is more recent than the last RUNSTATS, you might need to run RUNSTATS: `(JULIAN_DAY(REORGLASTTIME)>JULIAN_DAY(STATSLASTTIME))`
STATSINSERTS	INTEGER	The number of records or LOBs that have been inserted since the last RUNSTATS on the table space or partition. A null value means that the number of inserted records or LOBs is unknown.
STATSDELETES	INTEGER	The number of records or LOBs that have been deleted since the last RUNSTATS on the table space or partition. A null value means that the number of deleted records or LOBs is unknown.
STATSUPDATES	INTEGER	The number of rows that have been updated since the last RUNSTATS on the table space or partition. This value does not include LOB updates because LOB updates are really deletions followed by insertions. A null value means that the number of updated rows is unknown. This value can be used with STATSDELETES and STATSINSERTS to determine if RUNSTATS is necessary. For example, suppose that your site's maintenance policies require that RUNSTATS is to be run after 20% of the rows in a table space have changed. To determine if RUNSTATS is required, calculate the sum of updated, inserted, and deleted rows since the last RUNSTATS. Then calculate the ratio of that sum to the total number of rows. If the percentage is greater than 20, you need to run RUNSTATS: `(((STATSINSERTS+STATSDELETES+STATSUPDATES)*100)/TOTALROWS)>20`

Table 181. Descriptions of columns in the TABLESPACESTATS table (continued)

Column name	Data type	Description
STATSMASSDELETE	INTEGER	The number of mass deletes from a segmented or LOB table space, or the number of dropped tables from a segmented table space, since the last RUNSTATS. A null value means that the number of mass deletes is unknown. If this value is non-zero, RUNSTATS might be necessary.
COPYLASTTIME	TIMESTAMP	The timestamp of the last full or incremental image copy on the table space or partition. A null value means that COPY has never been run on the table space or partition, or that the timestamp of the last full image copy is unknown. You can compare this timestamp to the timestamp of the last REORG on the same object to determine when a COPY is needed. If the date of the last REORG is more recent than the last COPY, you might need to run COPY: `(JULIAN_DAY(REORGRLASTTIME)>JULIAN_DAY(COPYLASTTIME))`

COPYUPDATEDPAGES	INTEGER	The number of distinct pages that have been updated since the last COPY.
		
		A null value means that the number of updated pages is unknown.
		
		You can compare this value to the total number of pages to determine when a COPY is needed.
		
		For example, you might want to take an incremental image copy when 1% of the pages have changed:
		`((COPYUPDATEDPAGES*100)/NACTIVE)>1`
		
		You might want to take a full image copy when 20% of the pages have changed:
		`((COPYUPDATEDPAGES*100)/NACTIVE)>20`

COPYCHANGES	INTEGER	The number of insert, delete, and update operations since the last COPY.
		
		A null value means that the number of insert, delete, or update operations is unknown.
		
		This number indicates the approximate number of log records that DB2 needs to process to recover to the current state.
		
		For example, you might want to take an incremental image copy when DB2 processes more than 1% of the rows from the logs:
		`((COPYCHANGES*100)/TOTALROWS)>1`
		
		You might want to take a full image copy when DB2 processes more than 10% of the rows from the logs:
		`((COPYCHANGES*100)/TOTALROWS)>10`

Table 181. Descriptions of columns in the TABLESPACESTATS table (continued)

Column name	Data type	Description
COPYUPDATELRSN	CHAR(6) FOR BIT DATA	The LRSN or RBA of the first update after the last COPY. A null value means that the LRSN or RBA is unknown. Consider running COPY if this value is not in the active logs. To determine the oldest LRSN or RBA in the active logs, use the print log map utility (DSNJU004).
COPYUPDATETIME	TIMESTAMP	The timestamp of the first update after the last COPY. A null value means that the timestamp is unknown.

Table 182 describes the columns of the INDEXSPACESTATS table and explains how you can use them in deciding when to run REORG, RUNSTATS, or COPY.

Table 182. Descriptions of columns in the INDEXSPACESTATS table

Column name	Data type	Description
DBNAME	CHAR(8) NOT NULL	The name of the database. This column is used to map a database to its statistics.
NAME	CHAR(8) NOT NULL	The name of the index space. This column is used to map an index space to its statistics.
PARTITION	SMALLINT NOT NULL	This column is used to map a data set number in an index space to its statistics. The data set number within the index space. For partitioned index spaces, this value corresponds to the partition number for a single partition. For nonpartitioned index spaces, this value is 0.
DBID	SMALLINT NOT NULL	The internal identifier of the database. This column is used to map a DBID to its statistics.
ISOBID	SMALLINT NOT NULL	The internal identifier of the index space page set descriptor. This column is used to map an ISOBID to its statistics.
PSID	SMALLINT NOT NULL	The internal identifier of the table space page set descriptor for the table space associated with the index that is represented by this row. This column is used to map a PSID to the statistics for the associated index.

Table 182. Descriptions of columns in the INDEXSPACESTATS table (continued)

Column name	Data type	Description
UPDATESTATSTIME	TIMESTAMP NOT NULL WITH DEFAULT	The timestamp when the row was inserted or last updated. This column is updated with the current timestamp when a row in the INDEXSPACESTATS table is inserted or updated. You can use this column in several ways: • To determine the actions that caused the latest change to the INDEXSPACESTATS table. Do this by selecting any of the timestamp columns and comparing them to the UPDATESTATSTIME column. • To determine whether an analysis of data is needed. This determination might be based on a given time interval, or on a combination of the time interval and the amount of activity. For example, suppose that you want to analyze statistics for the last seven days. To determine whether any activity has occurred in the past seven days, check whether the difference between the current date and the UPDATESTATSTIME value is less than or equal to seven: `(JULIAN_DAY(CURRENT DATE)-JULIAN_DAY(UPDATESTATSTIME))<= 7`
TOTALENTRIES	FLOAT	The number of entries, including duplicate entries, in the index space or partition. A null value means that the number of entries is unknown, or that REORG, LOAD, or REBUILD has never been run. Use this value with the value of any column that contains a number of affected index entries to determine the percentage of index entries that are affected by a particular action.
NLEVELS	SMALLINT	The number of levels in the index tree. A null value means that the number of levels is unknown.
NACTIVE	INTEGER	The number of active pages in the index space or partition. This value is equivalent to the number of preformatted pages. A null value means that the number of active pages is unknown. Use this value with the value of any column that contains a number of affected pages to determine the percentage of pages that are affected by a particular action. For example, suppose that your site's maintenance policies require that COPY is to be run after 20% of the pages in an index space have changed. To determine if a COPY is required, calculate the ratio of updated pages since the last COPY to the total number of active pages. If the percentage is greater than 20, you need to run COPY: `((COPYUPDATEDPAGES*100)/NACTIVE)>20`
SPACE	INTEGER	The amount of space, in KB, that is allocated to the index space or partition. For multi-piece linear page sets, this value is the amount of space in all data sets. A null value means that the amount of space is unknown. Use this value to monitor growth and validate design assumptions.

Table 182. Descriptions of columns in the INDEXSPACESTATS table (continued)

Column name	Data type	Description
EXTENTS	SMALLINT	The number of extents in the index space or partition. For multi-piece index spaces, this value is the number of extents for the last data set. For a data set that is striped across multiple volumes, the value is the number of logical extents. A null value means that the number of extents is unknown. Use this value to determine: • When the primary allocation value for an index space or partition needs to be altered. • When you are approaching the maximum number of extents and risking extend failures.
LOADRLASTTIME	TIMESTAMP	The timestamp of the last LOAD REPLACE on the index space or partition. A null value means that the timestamp of the last LOAD REPLACE is unknown. If COPY YES was specified when the index was created (the value of COPY is Y in SYSIBM.SYSINDEXES), you can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last LOAD REPLACE is more recent than the last COPY, you might need to run COPY: `(JULIAN_DAY(LOADRLASTTIME)>JULIAN_DAY(COPYLASTTIME))`
REBUILDLASTTIME	TIMESTAMP	The timestamp of the last REBUILD INDEX on the index space or partition. A null value means that the timestamp of the last REBUILD INDEX is unknown. If COPY YES was specified when the index was created (the value of COPY is Y in SYSIBM.SYSINDEXES), you can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last REBUILD INDEX is more recent than the last COPY, you might need to run COPY: `(JULIAN_DAY(REBUILDLASTTIME)>JULIAN_DAY(COPYLASTTIME))`
REORGLASTTIME	TIMESTAMP	The timestamp of the last REORG INDEX on the index space or partition. A null value means that the timestamp of the last REORG INDEX is unknown. If COPY YES was specified when the index was created (the value of COPY is Y in SYSIBM.SYSINDEXES), you can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last REORG INDEX is more recent than the last COPY, you might need to run COPY: `(JULIAN_DAY(REORGLASTTIME)>JULIAN_DAY(COPYLASTTIME))`

Table 182. Descriptions of columns in the INDEXSPACESTATS table (continued)

Column name	Data type	Description
REORGINSERTS	INTEGER	The number of index entries that have been inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE on the index space or partition. A null value means that the number of inserted index entries is unknown.
REORGDELETES	INTEGER	The number of index entries that have been deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE on the index space or partition. A null value means that the number of deleted index entries is unknown. This value can be used with REORGINSERTS to determine if a REORG is necessary. For example, suppose that your site's maintenance policies require that REORG is to be run after 20% of the index entries have changed. To determine if a REORG is required, calculate the sum of inserted and deleted rows since the last REORG. Then calculate the ratio of that sum to the total number of index entries. If the percentage is greater than 20, you need to run REORG: `(((REORGINSERTS+REORGDELETES)*100)/TOTALENTRIES)>20`
REORGAPPENDINSERT	INTEGER	The number of index entries that have been inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE on the index space or partition that have a key value that is greater than the maximum key value in the index or partition. A null value means that the number of inserted index entries is unknown. This value can be used with REORGINSERTS to decide when to adjust the PCTFREE specification for the index. For example, if the ratio of REORGAPPENDINSERT to REORGINSERTS is greater than 10%, you might need to run ALTER INDEX to adjust PCTFREE or to run REORG more frequently: `((REORGAPPENDINSERT*100)/REORGINSERTS)>10`
REORGPSEUDODELETES	INTEGER	The number of index entries that have been pseudo-deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE on the index space or partition. A pseudo-delete is a RID entry that has been marked as deleted. A null value means that the number of pseudo-deleted index entries is unknown. This value can be used to determine if a REORG is necessary. For example, if the ratio of pseudo-deletes to total index entries is greater than 10%, you might need to run REORG: `((REORGPSEUDODELETES*100)/TOTALENTRIES)>10`
REORGMASSDELETE	INTEGER	The number of times that an index or index space partition was mass deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE. A null value means that the number of mass deletes is unknown. If this value is non-zero, a REORG might be necessary.

Table 182. Descriptions of columns in the INDEXSPACESTATS table (continued)

Column name	Data type	Description
REORGLEAFNEAR	INTEGER	The net number of leaf pages located physically near previous pages for successive active leaf pages that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE. The distance between leaf pages is optimal if the difference is 1 and considered near if the distance is 2-16. An index page is added during a page split and the distance between the predecessor and successor pages can decrement this count if the distance between the two was near. The distance between the predecessor and new page increment the count if they are near. The distance between the new page and successor increment the count if they are near. If a leaf page is deleted the distance between the new predecessor and successor pages can increment this count if the distance between the two is near. The distance between the predecessor and the deleted page decrement the count if it was near. The distance between the successor and the deleted page decrement the count if it was near. A null value means that the value is unknown. A negative value is possible in some cases.
REORGLEAFFAR	INTEGER	The net number of leaf pages located physically far away from previous leaf pages for successive active leaf pages that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE. The distance between leaf pages is optimal if the difference is 1 and considered far if the distance is greater than 16. An index page is added during a page split and the distance between the predecessor and successor pages can decrement this count if the distance between the two was far. The distance between the predecessor and new page increment the count if they are far. The distance between the new page and successor increment the count if they are far. If a leaf page is deleted the distance between the new predecessor and successor pages can increment this count if the distance between the two is far. The distance between the predecessor and the deleted page decrement the count if it was far. The distance between the successor and the deleted page decrement the count if it was far. A null value means that the value is unknown. This value can be used to decide when to run REORG. For example, calculate the ratio of relocated far leaf pages to the number of active pages. If this value is greater than 10 percent, you might need to run REORG: `((REORGLEAFFAR*100)/NACTIVE)>10`

Table 182. Descriptions of columns in the INDEXSPACESTATS table (continued)

Column name	Data type	Description
REORGNUMLEVELS	INTEGER	The number of levels in the index tree that were added or removed since the last REORG, REBUILD INDEX, or LOAD REPLACE. A null value means that the number of added or deleted levels is unknown. If this value has increased since the last REORG, REBUILD INDEX, or LOAD REPLACE, you need to check other values such as REORGPSEUDODELETES to determine whether to run REORG. If this value is less than zero, the index space contains empty pages. Running REORG can save disk space and decrease index sequential scan I/O time by eliminating those empty pages.
STATSLASTTIME	TIMESTAMP	The timestamp of the last RUNSTATS on the index space or partition. A null value means that RUNSTATS has never been run on the index space or partition, or that the timestamp of the last RUNSTATS is unknown. You can compare this timestamp to the timestamp of the last REORG on the same object to determine when RUNSTATS is needed. If the date of the last REORG is more recent than the last RUNSTATS, you might need to run RUNSTATS: `(JULIAN_DAY(REORGLASTTIME)>JULIAN_DAY(STATSLASTTIME))`
STATSINSERTS	INTEGER	The number of index entries that have been inserted since the last RUNSTATS on the index space or partition. A null value means that the number of inserted index entries unknown.
STATSDELETES	INTEGER	The number of index entries that have been deleted since the last RUNSTATS on the index space or partition. A null value means that the number of deleted index entries is unknown. This value can be used with STATSINSERTS to determine if RUNSTATS is necessary. For example, suppose that your site's maintenance policies require that RUNSTATS is run after 20% of the rows in an index space have changed. To determine if RUNSTATS is required, calculate the sum of inserted and deleted index entries since the last RUNSTATS. Then calculate the ratio of that sum to the total number of index entries. If the percentage is greater than 20, you need to run RUNSTATS: `(((STATSINSERTS+STATSDELETES)*100)/TOTALENTRIES)>20`
STATSMASSDELETE	INTEGER	The number of times that the index or index space partition was mass deleted since the last RUNSTATS. A null value means that the number of mass deletes is unknown. If this value is non-zero, RUNSTATS might be necessary.

Table 182. Descriptions of columns in the INDEXSPACESTATS table (continued)

Column name	Data type	Description
COPYLASTTIME	TIMESTAMP	The timestamp of the last full image copy on the index space or partition. A null value means that COPY has never been run on the index space or partition, or that the timestamp of the last full image copy is unknown. You can compare this timestamp to the timestamp of the last REORG on the same object to determine when a COPY is needed. If the date of the last REORG is more recent than the last COPY, you might need to run COPY: `(JULIAN_DAY(REORGRLASTTIME)>JULIAN_DAY(COPYLASTTIME))`
COPYUPDATEDPAGES	INTEGER	The number of distinct pages that have been updated since the last COPY. A null value means that the number of updated pages is unknown, or that the index was created with COPY NO. You can compare this value to the total number of pages to determine when a COPY is needed. For example, you might want to take a full image copy when 20% of the pages have changed: `((COPYUPDATEDPAGES*100)/NACTIVE)>20`
COPYCHANGES	INTEGER	The number of insert or delete operations since the last COPY. A null value means that the number of insert or update operations is unknown, or that the index was created with COPY NO. This number indicates the approximate number of log records that DB2 needs to process to recover to the current state. For example, you might want to take a full image copy when DB2 processes more than 10% of the index entries from the logs: `((COPYCHANGES*100)/TOTALENTRIES)>10`
COPYUPDATELRSN	CHAR(6) FOR BIT DATA	The LRSN or RBA of the first update after the last COPY. A null value means that the LRSN or RBA is unknown, or that the index was created with COPY NO. Consider running COPY if this value is not in the active logs. To determine the oldest LRSN or RBA in the active logs, use the print log map utility (DSNJU004).
COPYUPDATETIME	TIMESTAMP	The timestamp of the first update after the last COPY. A null value means that the timestamp is unknown, or tthat he index was created with COPY NO.

Operating with real-time statistics

To use the real-time statistics effectively, you need to understand when DB2 collects and externalizes them, and what factors in your system can affect the statistics. This section contains the following topics:

• "When DB2 externalizes real-time statistics" on page 888

- "How DB2 utilities affect the real-time statistics"
- "How non-DB2 utilities affect real-time statistics" on page 895
- "Real-time statistics on objects in work file databases and the TEMP database" on page 896
- "Real-time statistics on read-only or nonmodified objects" on page 896
- "How dropping objects affects real-time statistics" on page 896
- "How SQL operations affect real-time statistics counters" on page 896
- "Real-time statistics in data sharing" on page 897
- "Improving concurrency with real-time statistics" on page 898
- "Recovering the real-time statistics tables" on page 898
- "Statistics accuracy" on page 898

When DB2 externalizes real-time statistics

DB2 externalizes real-time statistics in the following circumstances:

- When you issue STOP DATABASE(*database-name*) SPACENAM(*space-name*)

  # This command externalizes statistics only for *database-name* and *space-name*. No
  # statistics are externalized when the DSNRTSDB database is stopped.
- At the end of the time interval that you specify during installation

  See "Setting the interval for writing real-time statistics" on page 875 for information about how to set this time interval.
- When you issue STOP DB2 MODE(QUIESCE)

  DB2 writes any statistics that are in memory when you issue this command to the statistics tables. However, if you issue STOP DB2 MODE(FORCE), DB2 does not write the statistics, and you lose them.
- During utility operations

  "How DB2 utilities affect the real-time statistics" gives details on how the utilities modify the statistics tables.

  DB2 does not maintain real-time statistics for any objects in the real-time statistics database. Therefore, if you run a utility with a utility list, and the list contains any real-time statistics objects, DB2 does not externalize real-time statistics during the execution of that utility for any of the objects in the utility
  | list. DB2 does not maintain interval counter real-time statistics for SYSLGRNG
  | and its indexes during utility operation. DB2 maintains statistics for these objects
  | only during non-utility operation.

  Recommendation: Do not include real-time statistics objects in utility lists.

DB2 does not externalize real-time statistics at a tracker site.

How DB2 utilities affect the real-time statistics

In general, SQL INSERT, UPDATE, and DELETE statements cause DB2 to modify the real-time statistics. However, certain DB2 utilities also affect the statistics. The following sections discuss the effect of each of those utilities on the statistics:

- "How LOAD affects real-time statistics" on page 889
- "How REORG affects real-time statistics" on page 891
- "How REBUILD INDEX affects real-time statistics" on page 893
- "How RUNSTATS affects real-time statistics" on page 893
- "How COPY affects real-time statistics" on page 894
- "How RECOVER affects real-time statistics" on page 895

How LOAD affects real-time statistics

Table 183 shows how running LOAD REPLACE on a table space or table space partition affects the TABLESPACESTATS statistics.

Table 183. Changed TABLESPACESTATS values during LOAD

Column name	Settings for LOAD REPLACE after RELOAD phase
TOTALROWS	Number of loaded rows or LOBs[1]
NACTIVE	Actual value
SPACE	Actual value
EXTENTS	Actual value
LOADRLASTTIME	Current timestamp
REORGINSERTS	0
REORGDELETES	0
REORGUPDATES	0
REORGDISORGLOB	0
REORGUNCLUSTINS	0
REORGMASSDELETE	0
REORGNEARINDREF	0
REORGFARINDEF	0
STATSLASTTIME	Current timestamp[2]
STATSINSERTS	0[2]
STATSDELETES	0[2]
STATSUPDATES	0[2]
STATSMASSDELETE	0[2]
COPYLASTTIME	Current timestamp[3]
COPYUPDATEDPAGES	0[3]
COPYCHANGES	0[3]
COPYUPDATELRSN	Null[3]
COPYUPDATETIME	Null[3]

Notes:

1. Under certain conditions, such as a utility restart, the LOAD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null. Some rows that are loaded into a table space and are included in this value might later be removed during the index validation phase or the referential integrity check. DB2 includes counts of those removed records in the statistics that record deleted records.
2. DB2 sets this value only if the LOAD invocation includes the STATISTICS option.
3. DB2 sets this value only if the LOAD invocation includes the COPYDDN option.

Table 184 shows how running LOAD REPLACE affects the INDEXSPACESTATS statistics for an index space or physical index partition.

Table 184. Changed INDEXSPACESTATS values during LOAD REPLACE

Column name	Settings for LOAD REPLACE after BUILD phase
TOTALENTRIES	Number of index entries added[1]

Table 184. Changed INDEXSPACESTATS values during LOAD REPLACE (continued)

Column name	Settings for LOAD REPLACE after BUILD phase
NLEVELS	Actual value
NACTIVE	Actual value
SPACE	Actual value
EXTENTS	Actual value
LOADRLASTTIME	Current timestamp
REORGINSERTS	0
REORGDELETES	0
REORGAPPENDINSERT	0
REORGPSEUDODELETES	0
REORGMASSDELETE	0
REORGLEAFNEAR	0
REORGLEAFFAR	0
REORGNUMLEVELS	0
STATSLASTTIME	Current timestamp[2]
STATSINSERTS	0[2]
STATSDELETES	0[2]
STATSMASSDELETE	0[2]
COPYLASTTIME	Current timestamp[3]
COPYUPDATEDPAGES	0[3]
COPYCHANGES	0[3]
COPYUPDATELRSN	Null[3]
COPYUPDATETIME	Null[3]

Notes:

1. Under certain conditions, such as a utility restart, the LOAD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.
2. DB2 sets this value only if the LOAD invocation includes the STATISTICS option.
3. DB2 sets this value only if the LOAD invocation includes the COPYDDN option.

For a logical index partition:

- A LOAD operation without the REPLACE option behaves similar to a SQL INSERT operation in that the number of records loaded are counted in the incremental counters such as REORGINSERTS, REORGAPPENDINSERT, STATSINSERTS, and COPYCHANGES. A LOAD operation without the REPLACE option affects the organization of the data and can be a trigger to run REORG, RUNSTATS or COPY.
- DB2 does not reset the nonpartitioned index when it does a LOAD REPLACE on a partition. Therefore, DB2 does not reset the statistics for the index. The REORG counters from the last REORG are still correct. DB2 updates LOADRLASTTIME when the entire nonpartitioned index is replaced.
- When DB2 does a LOAD RESUME YES on a partition, after the BUILD phase, DB2 increments TOTALENTRIES by the number of index entries that were inserted during the BUILD phase.

How REORG affects real-time statistics

Table 185 shows how running REORG on a table space or table space partition affects the TABLESPACESTATS statistics.

Table 185. Changed TABLESPACESTATS values during REORG

Column name	Settings for REORG SHRLEVEL NONE after RELOAD phase	Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase
TOTALROWS	Number rows or LOBs loaded[1]	For SHRLEVEL REFERENCE: Number of loaded rows or LOBs during RELOAD phase For SHRLEVEL CHANGE: Number of loaded rows or LOBs during RELOAD phase plus number of rows inserted during LOG APPLY phase minus number of rows deleted during LOG phase
NACTIVE	Actual value	Actual value
SPACE	Actual value	Actual value
EXTENTS	Actual value	Actual value
REORGLASTTIME	Current timestamp	Current timestamp
REORGINSERTS	0	Actual value[2]
REORGDELETES	0	Actual value[2]
REORGUPDATES	0	Actual value[2]
REORGDISORGLOB	0	Actual value[2]
REORGUNCLUSTINS	0	Actual value[2]
REORGMASSDELETE	0	Actual value[2]
REORGNEARINDREF	0	Actual value[2]
REORGFARINDEF	0	Actual value[2]
STATSLASTTIME	Current timestamp[3]	Current timestamp[3]
STATSINSERTS	0[3]	Actual value[2]
STATSDELETES	0[3]	Actual value[2]
STATSUPDATES	0[3]	Actual value[2]
STATSMASSDELETE	0[3]	Actual value[2]
COPYLASTTIME	Current timestamp[4]	Current timestamp
COPYUPDATEDPAGES	0[4]	Actual value[2]
COPYCHANGES	0[4]	Actual value[2]
COPYUPDATELRSN	Null[4]	Actual value[5]
COPYUPDATETIME	Null[4]	Actual value[5]

Notes:

1. Under certain conditions, such as a utility restart, the REORG utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null. Some rows that are loaded into a table space and are included in this value might later be removed during the index validation phase or the referential integrity check. DB2 includes counts of those removed records in the statistics that record deleted records.
2. This is the actual number of inserts, updates, or deletes that are due to applying the log to the shadow copy.
3. DB2 sets this value only if the REORG invocation includes the STATISTICS option.
4. DB2 sets this value only if the REORG invocation includes the COPYDDN option.
5. This is the LRSN or timestamp for the first update that is due to applying the log to the shadow copy.

Table 186 shows how running REORG affects the INDEXSPACESTATS statistics for an index space or physical index partition.

Table 186. Changed INDEXSPACESTATS values during REORG

Column name	Settings for REORG SHRLEVEL NONE after RELOAD phase	Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase
TOTALENTRIES	Number of index entries added[1]	For SHRLEVEL REFERENCE: Number of added index entries during BUILD phase For SHRLEVEL CHANGE: Number of added index entries during BUILD phase plus number of added index entries during LOG phase minus number of deleted index entries during LOG phase
NLEVELS	Actual value	Actual value
NACTIVE	Actual value	Actual value
SPACE	Actual value	Actual value
EXTENTS	Actual value	Actual value
REORGLASTTIME	Current timestamp	Current timestamp
REORGINSERTS	0	Actual value[2]
REORGDELETES	0	Actual value[2]
REORGAPPENDINSERT	0	Actual value[2]
REORGPSEUDODELETES	0	Actual value[2]
REORGMASSDELETE	0	Actual value[2]
REORGLEAFNEAR	0	Actual value[2]
REORGLEAFFAR	0	Actual value[2]
REORGNUMLEVELS	0	Actual value[2]
STATSLASTTIME	Current timestamp[3]	Current timestamp[3]
STATSINSERTS	0[3]	Actual value[2]
STATSDELETES	0[3]	Actual value[2]
STATSMASSDELETE	0[3]	Actual value[2]
COPYLASTTIME	Current timestamp[4]	Unchanged[5]
COPYUPDATEDPAGES	0[4]	Unchanged[5]
COPYCHANGES	0[4]	Unchanged[5]
COPYUPDATELRSN	Null[4]	Unchanged[5]
COPYUPDATETIME	Null[4]	Unchanged[5]

Notes:

1. Under certain conditions, such as a utility restart, the REORG utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.
2. This is the actual number of inserts, updates, or deletes that are due to applying the log to the shadow copy.
3. DB2 sets this value only if the REORG invocation includes the STATISTICS option.
4. DB2 sets this value only if the REORG invocation includes the COPYDDN option.
5. Inline COPY is not allowed for SHRLEVEL CHANGE or SHRLEVEL REFERENCE.

For a logical index partition, DB2 does not reset the nonpartitioned index when it does a REORG on a partition. Therefore, DB2 does not reset the statistics for the index. The REORG counters and REORGLASTTIME are relative to the last time the entire nonpartitioned index is reorganized. In addition, the REORG counters might be low because, due to the methodology, some index entries are changed during REORG of a partition.

How REBUILD INDEX affects real-time statistics

Table 187 shows how running REBUILD INDEX affects the INDEXSPACESTATS statistics for an index space or physical index partition.

Table 187. Changed INDEXSPACESTATS values during REBUILD INDEX

Column name	Settings after BUILD phase
TOTALENTRIES	Number of index entries added[1]
NLEVELS	Actual value
NACTIVE	Actual value
SPACE	Actual value
EXTENTS	Actual value
REBUILDLASTTIME	Current timestamp
REORGINSERTS	0
REORGDELETES	0
REORGAPPENDINSERT	0
REORGPSEUDODELETES	0
REORGMASSDELETE	0
REORGLEAFNEAR	0
REORGLEAFFAR	0
REORGNUMLEVELS	0

Note:

1. Under certain conditions, such as a utility restart, the REBUILD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.

For a logical index partition, DB2 does not collect TOTALENTRIES statistics for the entire nonpartitioned index when it runs REBUILD INDEX. Therefore, DB2 does not reset the statistics for the index. The REORG counters from the last REORG are still correct. DB2 updates REBUILDLASTTIME when the entire nonpartitioned index is rebuilt.

How RUNSTATS affects real-time statistics

Only RUNSTATS UPDATE ALL affects the real-time statistics. When the RUNSTATS job starts, DB2 externalizes all in-memory statistics to the real-time statistics tables.

Table 188 shows how running RUNSTATS UPDATE ALL on a table space or table space partition affects the TABLESPACESTATS statistics.

Table 188. Changed TABLESPACESTATS values during RUNSTATS UPDATE ALL

Column name	During UTILINIT phase	After RUNSTATS phase
STATSLASTTIME	Current timestamp[1]	Timestamp of the start of RUNSTATS phase

Table 188. Changed TABLESPACESTATS values during RUNSTATS UPDATE ALL (continued)

Column name	During UTILINIT phase	After RUNSTATS phase
STATSINSERTS	Actual value[1]	Actual value[2]
STATSDELETES	Actual value[1]	Actual value[2]
STATSUPDATES	Actual value[1]	Actual value[2]
STATSMASSDELETE	Actual value[1]	Actual value[2]

Notes:

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

Table 189 shows how running RUNSTATS UPDATE ALL on an index affects the INDEXSPACESTATS statistics.

Table 189. Changed INDEXSPACESTATS values during RUNSTATS UPDATE ALL

Column name	During UTILINIT phase	After RUNSTATS phase
STATSLASTTIME	Current timestamp[1]	Timestamp of the start of RUNSTATS phase
STATSINSERTS	Actual value[1]	Actual value[2]
STATSDELETES	Actual value[1]	Actual value[2]
STATSMASSDELETE	Actual value[1]	Actual value[2]

Notes:

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

How COPY affects real-time statistics

When a COPY job starts, DB2 externalizes all in-memory statistics to the real-time statistics tables. Statistics are gathered for a full image copy or an incremental copy, but not for a data set copy.

Table 190 shows how running COPY on a table space or table space partition affects the TABLESPACESTATS statistics.

Table 190. Changed TABLESPACESTATS values during COPY

Column name	During UTILINIT phase	After COPY phase
COPYLASTTIME	Current timestamp[1]	Timestamp of the start of COPY phase
COPYUPDATEDPAGES	Actual value[1]	Actual value[2]
COPYCHANGES	Actual value[1]	Actual value[2]
COPYUPDATELRSN	Actual value[1]	Actual value[3]
COPYUPDATETIME	Actual value[1]	Actual value[3]

Table 190. Changed TABLESPACESTATS values during COPY (continued)

Column name	During UTILINIT phase	After COPY phase

Notes:

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.
3. This value is null for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

Table 191 shows how running COPY on an index affects the INDEXSPACESTATS statistics.

Table 191. Changed INDEXSPACESTATS values during COPY

Column name	During UTILINIT phase	After COPY phase
COPYLASTTIME	Current timestamp[1]	Timestamp of the start of COPY phase
COPYUPDATEDPAGES	Actual value[1]	Actual value[2]
COPYCHANGES	Actual value[1]	Actual value[2]
COPYUPDATELRSN	Actual value[1]	Actual value[3]
COPYUPDATETIME	Actual value[1]	Actual value[3]

Note:

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.
3. This value is null for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

How RECOVER affects real-time statistics

After recovery to the current state, the in-memory counter fields are still valid, so DB2 does not modify them. However, after a point-in-time recovery, the statistics might not be valid. DB2 therefore sets all the REORG, STATS, and COPY counter statistics to null after a point-in-time recovery. After recovery to the current state, DB2 sets NACTIVE, SPACE, and EXTENTS to their new values. After a point-in-time recovery, DB2 sets NLEVELS, NACTIVE, SPACE, and EXTENTS to their new values.

How non-DB2 utilities affect real-time statistics

Non-DB2 utilities do not affect real-time statistics. Therefore, an object that is the target of a non-DB2 COPY, LOAD, REBUILD, REORG, or RUNSTATS job can cause incorrect statistics to be inserted in the real-time statistics tables. Follow this process to ensure correct statistics when you run non-DB2 utilities:

1. Stop the table space or index on which you plan to run the utility. This action causes DB2 to write the in-memory statistics to the real-time statistics tables and initialize the in-memory counters. If DB2 cannot externalize the statistics, the STOP command does not fail.
2. Run the utility.
3. When the utility completes, update the statistics tables with new totals and timestamps, and put zero values in the incremental counter.

Real-time statistics on objects in work file databases and the TEMP database

Although you cannot run utilities on objects in the work files databases and TEMP database, DB2 records the NACTIVE, SPACE, and EXTENTS statistics on table spaces in those databases.

Real-time statistics for DEFINE NO objects

For objects that are created with DEFINE NO, no row is inserted into the real-time statistics table until the object is physically defined.

Real-time statistics on read-only or nonmodified objects

DB2 does not externalize the NACTIVE, SPACE, or EXTENTS statistics for read-only objects or objects that are not modified.

How dropping objects affects real-time statistics

If you drop a table space or index, DB2 deletes its statistics from the real-time statistics tables. However, if the real-time statistics database is not available when you drop a table space or index, the statistics remain in the real-time statistics tables, even though the corresponding object no longer exists. You need to use SQL DELETE statements to manually remove those rows from the real-time statistics tables.

If a row still exists in the real-time statistics tables for a dropped table space or index, and if you create a new object with the same DBID and PSID as the dropped object, DB2 reinitializes the row before it updates any values in that row.

How SQL operations affect real-time statistics counters

SQL operations affect the counter columns in the real-time statistics tables. These are the columns that record the number of insert, delete, or update operations, as well as the total counters, TOTALROWS, and TOTALENTRIES.

UPDATE: When you perform an UPDATE, DB2 increments the update counters.

INSERT: When you perform an INSERT, DB2 increments the insert counters. DB2 keeps separate counters for clustered and unclustered INSERTs.

DELETE: When you perform a DELETE, DB2 increments the delete counters.

ROLLBACK: When you perform a ROLLBACK, DB2 increments the counters, depending on the type of SQL operation that is rolled back:

Rolled-back SQL statement	Incremented counters
UPDATE	Update counters
INSERT	Delete counters
DELETE	Insert counters

Notice that for INSERT and DELETE, the counter for the inverse operation is incremented. For example, if two INSERT statements are rolled back, the delete counter is incremented by 2.

UPDATE of partitioning keys: If an update to a partitioning key causes rows to move to a new partition, the following real-time statistics are impacted:

Action	Incremented counters
When UPDATE is executed	Update count of old partition = +1 Insert count of new partition = +1
When UPDATE is committed	Delete count of old partition = +1
When UPDATE is rolled back	Update count of old partition = +1 (compensation log record) Delete count of new partition = +1 (remove inserted record)

If an update to a partitioning key does not cause rows to move to a new partition, the counts are accumulated as expected:

Action	Incremented counters
When UPDATE is executed	Update count of current partition = +1 NEAR/FAR indirect reference count = +1 (if overflow occured)
When UPDATE is rolled back	Update count of current partition = +1 (compensation log record)

Mass DELETE:Performing a mass delete operation on a table space does not cause DB2 to reset the counter columns in the real-time statistics tables. After a mass delete operation, the value in a counter column includes the count from a time prior to the mass delete operation, as well as the count after the mass delete operation.

Real-time statistics in data sharing

In a data sharing environment, DB2 members update their statistics serially. Each member reads the target row from the statistics table, obtains a lock, aggregates its in-memory statistics, and updates the statistics table with the new totals. Each member sets its own interval for writing real-time statistics.

DB2 does locking based on the lock size of the DSNRTSDB.DSNRTSTS table space. DB2 uses cursor stability isolation and CURRENTDATA(YES) when it reads the statistics tables.

At the beginning of a RUNSTATS job, all data sharing members externalize their statistics to the real-time statistics tables and reset their in-memory statistics. If all members cannot externalize their statistics, DB2 sets STATSLASTTIME to null. An error in gathering and externalizing statistics does not prevent RUNSTATS from running.

At the beginning of a COPY job, all data sharing members externalize their statistics to the real-time statistics tables and reset their in-memory statistics. If all members cannot externalize their statistics, DB2 sets COPYLASTTIME to null. An error in gathering and externalizing statistics does not prevent COPY from running.

Utilities that reset page sets to empty can invalidate the in-memory statistics of other DB2 members. The member that resets a page set notifies the other DB2 members that a page set has been reset to empty, and the in-memory statistics are

invalidated. If the notify process fails, the utility that resets the page set does not fail. DB2 sets the appropriate timestamp (REORGLASTTIME, STATSLASTTIME, or COPYLASTTIME) to null in the row for the empty page set to indicate that the statistics for that page set are unknown.

Improving concurrency with real-time statistics

Follow these recommendations to reduce the risk of timeouts and deadlocks when you work with the real-time statistics tables:

- When you run COPY, RUNSTATS, or REORG on the real-time statistics objects, use SHRLEVEL CHANGE.
- When you execute SQL statements to query the real-time statistics tables, use uncommitted read isolation.

Recovering the real-time statistics tables

When you recover a DB2 subsystem after a disaster, DB2 starts with the ACESS(MAINT) option. No statistics are externalized in this state. Therefore, you need to perform the following actions on the real-time statistics database:

- Recover the real-time statistics objects after you recover the DB2 catalog and directory.
- Start the real-time statistics database explicitly, after DB2 restart.

Statistics accuracy

In general, the real-time statistics are accurate values. However, several factors can affect the accuracy of the statistics:

- Certain utility restart scenarios
- Certain utility operations that leave indexes in a database restrictive state, such as RECOVER-pending (RECP)

  Always consider the database restrictive state of objects before accepting a utility recommendation that is based on real-time statistics.
- A DB2 subsystem failure
- A notify failure in a data sharing environment

If you think that some statistics values might be inaccurate, you can correct the statistics by running REORG, RUNSTATS, or COPY on the objects for which DB2 generated the statistics.

Appendix F. Delimited file format

A delimited file is a sequential file with column delimiters. Each delimited file is a stream of records, which consists of fields that are ordered by column. Each record contains fields for one row. Within each row, individual fields are separated by column delimiters. All fields must be delimited character strings, non-delimited character strings, or external numeric values. Delimited character strings can contain column delimiters and can also contain character string delimiters when two successive character string delimiters are used to represent one character.

All characters in all records are in the same CCSID. If EBCDIC or ASCII data contains DBCS characters, the data must be in an appropriate mixed CCSID. If the data is Unicode it must be in CCSID 1208.

Figure 169 describes the format of delimited files that can be loaded into or unloaded from tables by using the LOAD and UNLOAD utilities.

```
Delimited file ::= Row 1 data ||
                   Row 2 data ||
                   .
                   .
                   .
                   Row n data

Row i data ::= Cell value(i,1) || Column delimiter ||
               Cell value(i,2) || Column delimiter ||
               .
               .
               .
               Cell value(i,m)

Column delimiter ::= Character specified by COLDEL option;
                       the default value is a comma (,)

Cell value(i,j) ::= Leading spaces ||
                    External numeric values ||
                    Delimited character string ||
                    Non-delimited character string ||
                    Trailing spaces

Non-delimited character string ::= A set of any characters except
                                     a column delimiter

Delimited character string ::= A character string delimiter ||
                               A set of any characters except a
                                 character string delimiter unless
                                 the character string delimiter is
                                 part of two successive character
                                 string delimiters ||
                               A character string delimiter ||
                               Trailing garbage

Character string delimiter ::= Character specified by CHARDEL option; the default
                                 value is a double quotation mark (")

Trailing garbage ::= A set of any characters except a column delimiter
```

Figure 169. Format of delimited files

For more information about the COLDEL and CHARDEL options for the LOAD utility, see "Option descriptions" on page 198. For more information about the COLDEL and CHARDEL options for the UNLOAD utility, see "Option descriptions" on page 615.

The following topics provide additional information:

- "Restrictions"
- "Delimited data types"
- "Examples of delimited files" on page 901

Restrictions

For delimiter restrictions, see "Loading delimited files" on page 244 or "Unloading delimited files" on page 656.

Delimited data types

Table 192 identifies the acceptable data type forms for the delimited file format that the LOAD and UNLOAD utilities use.

Table 192. Acceptable data type forms for delimited files

Data type	Acceptable form for loading a delimited file	Form that is created by unloading a delimited file
CHAR, VARCHAR	A delimited or non-delimited character string	Character data that is enclosed by character delimiters. For VARCHAR, length bytes do not precede the data in the string.
GRAPHIC (any type)[4]	A delimited or non-delimited character stream	Data that is unloaded as a delimited character string. For VARGRAPHIC, length bytes do not precede the data in the string.
INTEGER (any type)[1]	A stream of characters that represents a number in EXTERNAL format	Numeric data in external format.
DECIMAL (any type)[2]	A character string that represents a number in EXTERNAL format	A string of characters that represents a number.
FLOAT [3]	A representation of a number in the range `-7.2E+75` to `7.2E+75` in EXTERNAL format	A string of characters that represents a number in floating-point notation.
BLOB, CLOB	A delimited or non-delimited character string	Character data that is enclosed by character delimiters. Length bytes do not precede the data in the string.
DBCLOB	A delimited or non-delimited character string	Character data that is enclosed by character delimiters. Length bytes do not precede the data in the string.

Table 192. Acceptable data type forms for delimited files (continued)

Data type	Acceptable form for loading a delimited file	Form that is created by unloading a delimited file
DATE	A delimited or non-delimited character string that contains a date value in EXTERNAL format	Character string representation of a date.
TIME	A delimited or non-delimited character string that contains a time value in EXTERNAL format	Character string representation of a time.
TIMESTAMP	A delimited or non-delimited character string that contains a timestamp value in EXTERNAL format	Character string representation of a timestamp.

Notes:

1. Field specifications of INTEGER or SMALLINT are treated as INTEGER EXTERNAL.
2. Field specifications of DECIMAL, DECIMAL PACKED, or DECIMAL ZONED are treated as DECIMAL EXTERNAL.
3. Field specifications of FLOAT, REAL, or DOUBLE are treated as FLOAT EXTERNAL.
4. EBCID graphic data must be enclosed in shift-out and shift-in characters.

Examples of delimited files

Figure 170 shows an example of a delimited file with delimited character strings. In this example, the column delimiter is a comma (,). Because the character strings contain the column delimiter character, they must be delimited with character string delimiters. In this example, the character string delimiter is a double quotation mark (").

```
"Smith, Bob",4973,15.46
"Jones, Bill",12345,16.34
"Williams, Sam",452,193.78
```

Figure 170. Example of a delimited file with delimited character strings

Figure 171 shows an example of a delimited file with non-delimited character strings. In this example, the column delimiter is a semicolon (;). Because the character strings do not contain the column delimiter character, they do not need to be delimited with character string delimiters.

```
Smith, Bob;4973;15.46
Jones, Bill;12345;16.34
Williams, Sam;452;193.78
```

Figure 171. Example of a delimited file with non-delimited character strings

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*.

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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
3-2-12, Roppongi, Minato-ku
Tokyo 106-8711
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.

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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Programming interface information

This book is intended to help you to use DB2 UDB for z/OS utilities.

This book also documents General-use Programming Interface and Associated Guidance Information and Product-sensitive 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.

General-use Programming Interface and Associated Guidance Information is identified where it occurs, by an entry in a column of a table.

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of this IBM software product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may require changes in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs by an entry in a column of a table, or by the following marking:

Product-sensitive Programming Interface

Product-sensitive Programming Interface and Associated Guidance Information ...

End of Product-sensitive Programming Interface

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered marks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml.

Microsoft®, Windows®, Windows NT®, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX® is a registered trademark of The Open Group in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Glossary

The following terms and abbreviations are defined as they are used in the DB2 library.

A

abend. Abnormal end of task.

abend reason code. A 4-byte hexadecimal code that uniquely identifies a problem with DB2.

abnormal end of task (abend). Termination of a task, job, or subsystem because of an error condition that recovery facilities cannot resolve during execution.

access method services. The facility that is used to define and reproduce VSAM key-sequenced data sets.

access path. The path that is used to locate data that is specified in SQL statements. An access path can be indexed or sequential.

active log. The portion of the DB2 log to which log records are written as they are generated. The active log always contains the most recent log records, whereas the archive log holds those records that are older and no longer fit on the active log.

active member state. A state of a member of a data sharing group. The cross-system coupling facility identifies each active member with a group and associates the member with a particular task, address space, and z/OS system. A member that is not active has either a failed member state or a quiesced member state.

address space. A range of virtual storage pages that is identified by a number (ASID) and a collection of segment and page tables that map the virtual pages to real pages of the computer's memory.

address space connection. The result of connecting an allied address space to DB2. Each address space that contains a task that is connected to DB2 has exactly one address space connection, even though more than one task control block (TCB) can be present. See also *allied address space* and *task control block*.

| **address space identifier (ASID).** A unique
| system-assigned identifier for and address space.

administrative authority. A set of related privileges that DB2 defines. When you grant one of the administrative authorities to a person's ID, the person has all of the privileges that are associated with that administrative authority.

after trigger. A trigger that is defined with the trigger activation time AFTER.

agent. As used in DB2, the structure that associates all processes that are involved in a DB2 unit of work. An *allied agent* is generally synonymous with an *allied thread*. *System agents* are units of work that process tasks that are independent of the allied agent, such as prefetch processing, deferred writes, and service tasks.

aggregate function. An operation that derives its
result by using values from one or more rows. Contrast
with *scalar function*.

alias. An alternative name that can be used in SQL statements to refer to a table or view in the same or a remote DB2 subsystem.

allied address space. An area of storage that is external to DB2 and that is connected to DB2. An allied address space is capable of requesting DB2 services.

allied thread. A thread that originates at the local DB2 subsystem and that can access data at a remote DB2 subsystem.

allocated cursor. A cursor that is defined for stored procedure result sets by using the SQL ALLOCATE CURSOR statement.

already verified. An LU 6.2 security option that allows DB2 to provide the user's verified authorization ID when allocating a conversation. With this option, the user is not validated by the partner DB2 subsystem.

| **ambiguous cursor.** A database cursor that is in a plan
| or package that contains either PREPARE or EXECUTE
| IMMEDIATE SQL statements, and for which the
| following statements are true: the cursor is not defined
| with the FOR READ ONLY clause or the FOR UPDATE
| OF clause; the cursor is not defined on a read-only
| result table; the cursor is not the target of a WHERE
| CURRENT clause on an SQL UPDATE or DELETE
| statement.

American National Standards Institute (ANSI). An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

ANSI. American National Standards Institute.

APAR. Authorized program analysis report.

APAR fix corrective service. A temporary correction of an IBM software defect. The correction is temporary,

because it is usually replaced at a later date by a more permanent correction, such as a program temporary fix (PTF).

APF. Authorized program facility.

API. Application programming interface.

APPL. A VTAM® network definition statement that is used to define DB2 to VTAM as an application program that uses SNA LU 6.2 protocols.

application. A program or set of programs that performs a task; for example, a payroll application.

application-directed connection. A connection that an application manages using the SQL CONNECT statement.

application plan. The control structure that is produced during the bind process. DB2 uses the application plan to process SQL statements that it encounters during statement execution.

application process. The unit to which resources and locks are allocated. An application process involves the execution of one or more programs.

application programming interface (API). A functional interface that is supplied by the operating system or by a separately orderable licensed program that allows an application program that is written in a high-level language to use specific data or functions of the operating system or licensed program.

application requester. The component on a remote system that generates DRDA® requests for data on behalf of an application. An application requester accesses a DB2 database server using the DRDA application-directed protocol.

application server. The target of a request from a remote application. In the DB2 environment, the application server function is provided by the distributed data facility and is used to access DB2 data from remote applications.

archive log. The portion of the DB2 log that contains log records that have been copied from the active log.

ASCII. An encoding scheme that is used to represent strings in many environments, typically on PCs and workstations. Contrast with *EBCDIC* and *Unicode*.

ASID. Address space identifier.

attachment facility. An interface between DB2 and TSO, IMS, CICS, or batch address spaces. An attachment facility allows application programs to access DB2.

attribute. A characteristic of an entity. For example, in database design, the phone number of an employee is one of that employee's attributes.

authorization ID. A string that can be verified for connection to DB2 and to which a set of privileges is allowed. It can represent an individual, an organizational group, or a function, but DB2 does not determine this representation.

authorized program analysis report (APAR). A report of a problem that is caused by a suspected defect in a current release of an IBM supplied program.

authorized program facility (APF). A facility that permits the identification of programs that are authorized to use restricted functions.

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.

auxiliary index. An index on an auxiliary table in which each index entry refers to a LOB.

auxiliary table. A table that stores columns outside the table in which they are defined. Contrast with *base table*.

B

backout. The process of undoing uncommitted changes that an application process made. This might be necessary in the event of a failure on the part of an application process, or as a result of a deadlock situation.

backward log recovery. The fourth and final phase of restart processing during which DB2 scans the log in a backward direction to apply UNDO log records for all aborted changes.

base table. (1) A table that is created by the SQL CREATE TABLE statement and that holds persistent data. Contrast with *result table* and *temporary table*.

(2) A table containing a LOB column definition. The actual LOB column data is not stored with the base table. The base table contains a row identifier for each row and an indicator column for each of its LOB columns. Contrast with *auxiliary table*.

base table space. A table space that contains base tables.

basic predicate. A predicate that compares two values.

basic sequential access method (BSAM). An access method for storing or retrieving data blocks in a continuous sequence, using either a sequential-access or a direct-access device.

| **batch message processing program.** In IMS, an
| application program that can perform batch-type
| processing online and can access the IMS input and
| output message queues.

before trigger. A trigger that is defined with the trigger activation time BEFORE.

binary integer. A basic data type that can be further classified as small integer or large integer.

binary large object (BLOB). A sequence of bytes in
which the size of the value ranges from 0 bytes to
2 GB–1. Such a string has a CCSID value of 65535.

binary string. A sequence of bytes that is not associated with a CCSID. For example, the BLOB data type is a binary string.

bind. The process by which the output from the SQL precompiler is converted to a usable control structure, often called an access plan, application plan, or package. During this process, access paths to the data are selected and some authorization checking is performed. The types of bind are:

automatic bind. (More correctly, *automatic rebind*) A process by which SQL statements are bound automatically (without a user issuing a BIND command) when an application process begins execution and the bound application plan or package it requires is not valid.

dynamic bind. A process by which SQL statements are bound as they are entered.

incremental bind. A process by which SQL statements are bound during the execution of an application process.

static bind. A process by which SQL statements are bound after they have been precompiled. All static SQL statements are prepared for execution at the same time.

bit data. Data that is character type CHAR or
VARCHAR and has a CCSID value of 65535.

BLOB. Binary large object.

block fetch. A capability in which DB2 can retrieve, or fetch, a large set of rows together. Using block fetch can significantly reduce the number of messages that are being sent across the network. Block fetch applies only to cursors that do not update data.

BMP. Batch Message Processing (IMS). See *batch message processing program*.

bootstrap data set (BSDS). A VSAM data set that contains name and status information for DB2, as well as RBA range specifications, for all active and archive log data sets. It also contains passwords for the DB2 directory and catalog, and lists of conditional restart and checkpoint records.

BSAM. Basic sequential access method.

BSDS. Bootstrap data set.

buffer pool. Main storage that is reserved to satisfy the buffering requirements for one or more table spaces or indexes.

built-in data type. A data type that IBM supplies. Among the built-in data types for DB2 UDB for z/OS are string, numeric, ROWID, and datetime. Contrast with *distinct type*.

built-in function. A function that DB2 supplies. Contrast with *user-defined function*.

business dimension. A category of data, such as products or time periods, that an organization might want to analyze.

C

cache structure. A coupling facility structure that stores data that can be available to all members of a Sysplex. A DB2 data sharing group uses cache structures as group buffer pools.

CAF. Call attachment facility.

call attachment facility (CAF). A DB2 attachment facility for application programs that run in TSO or z/OS batch. The CAF is an alternative to the DSN command processor and provides greater control over the execution environment.

call-level interface (CLI). A callable application programming interface (API) for database access, which is an alternative to using embedded SQL. In contrast to embedded SQL, DB2 ODBC (which is based on the CLI architecture) does not require the user to precompile or bind applications, but instead provides a standard set of functions to process SQL statements and related services at run time.

cascade delete. The way in which DB2 enforces referential constraints when it deletes all descendent rows of a deleted parent row.

CASE expression. An expression that is selected based on the evaluation of one or more conditions.

cast function. A function that is used to convert instances of a (source) data type into instances of a different (target) data type. In general, a cast function has the name of the target data type. It has one single argument whose type is the source data type; its return type is the target data type.

castout. The DB2 process of writing changed pages from a group buffer pool to disk.

castout owner. The DB2 member that is responsible for casting out a particular page set or partition.

catalog. In DB2, a collection of tables that contains descriptions of objects such as tables, views, and indexes.

catalog table. Any table in the DB2 catalog.

CCSID. Coded character set identifier.

CDB. Communications database.

CDRA. Character Data Representation Architecture.

CEC. Central electronic complex. See *central processor complex*.

central electronic complex (CEC). See *central processor complex*.

central processor (CP). The part of the computer that contains the sequencing and processing facilities for instruction execution, initial program load, and other machine operations.

central processor complex (CPC). A physical collection of hardware (such as an ES/3090™) that consists of main storage, one or more central processors, timers, and channels.

| **CFRM.** Coupling facility resource management.

CFRM policy. A declaration by a z/OS administrator regarding the allocation rules for a coupling facility structure.

character conversion. The process of changing characters from one encoding scheme to another.

Character Data Representation Architecture (CDRA). An architecture that is used to achieve consistent representation, processing, and interchange of string data.

character large object (CLOB). A sequence of bytes representing single-byte characters or a mixture of single- and double-byte characters where the size of the value can be up to 2 GB–1. In general, character large object values are used whenever a character string might exceed the limits of the VARCHAR type.

character set. A defined set of characters.

character string. A sequence of bytes that represent bit data, single-byte characters, or a mixture of single-byte and multibyte characters.

check constraint. A user-defined constraint that specifies the values that specific columns of a base table can contain.

check integrity. The condition that exists when each row in a table conforms to the check constraints that are defined on that table. Maintaining check integrity requires DB2 to enforce check constraints on operations that add or change data.

| **check pending.** A state of a table space or partition that prevents its use by some utilities and by some SQL statements because of rows that violate referential constraints, check constraints, or both.

checkpoint. A point at which DB2 records internal status information on the DB2 log; the recovery process uses this information if DB2 abnormally terminates.

| **child lock.** For explicit hierarchical locking, a lock that is held on either a table, page, row, or a large object (LOB). Each child lock has a parent lock. See also *parent lock*.

CI. Control interval.

| **CICS.** Represents (in this publication): CICS Transaction Server for z/OS: Customer Information Control System Transaction Server for z/OS.

CICS attachment facility. A DB2 subcomponent that uses the z/OS subsystem interface (SSI) and cross-storage linkage to process requests from CICS to DB2 and to coordinate resource commitment.

CIDF. Control interval definition field.

claim. A notification to DB2 that an object is being accessed. Claims prevent drains from occurring until the claim is released, which usually occurs at a commit point. Contrast with *drain*.

claim class. A specific type of object access that can be one of the following isolation levels:
Cursor stability (CS)
Repeatable read (RR)
Write

claim count. A count of the number of agents that are accessing an object.

class of service. A VTAM term for a list of routes through a network, arranged in an order of preference for their use.

class word. A single word that indicates the nature of a data attribute. For example, the class word PROJ indicates that the attribute identifies a project.

clause. In SQL, a distinct part of a statement, such as a SELECT clause or a WHERE clause.

CLI. Call- level interface.

client. See *requester*.

CLIST. Command list. A language for performing TSO tasks.

CLOB. Character large object.

closed application. An application that requires exclusive use of certain statements on certain DB2

objects, so that the objects are managed solely through the application's external interface.

CLPA. Create link pack area.

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.

coded character set. A set of unambiguous rules that establish a character set and the one-to-one relationships between the characters of the set and their coded representations.

coded character set identifier (CCSID). A 16-bit number that uniquely identifies a coded representation of graphic characters. It designates an encoding scheme identifier and one or more pairs consisting of a character set identifier and an associated code page identifier.

code page. (1) A set of assignments of characters to code points. In EBCDIC, for example, the character 'A' is assigned code point X'C1' (2) , and character 'B' is assigned code point X'C2'. Within a code page, each code point has only one specific meaning.

code point. In CDRA, a unique bit pattern that represents a character in a code page.

code unit. The fundamental binary width in a computer architecture that is used for representing character data, such as 7 bits, 8 bits, 16 bits, or 32 bits. Depending on the character encoding form that is used, each code point in a coded character set can be represented internally by one or more code units.

coexistence. During migration, the period of time in which two releases exist in the same data sharing group.

cold start. A process by which DB2 restarts without processing any log records. Contrast with *warm start*.

collection. A group of packages that have the same qualifier.

column. The vertical component of a table. A column has a name and a particular data type (for example, character, decimal, or integer).

column function. See *aggregate function*.

"come from" checking. An LU 6.2 security option that defines a list of authorization IDs that are allowed to connect to DB2 from a partner LU.

command. A DB2 operator command or a DSN subcommand. A command is distinct from an SQL statement.

command prefix. A one- to eight-character command identifier. The command prefix distinguishes the command as belonging to an application or subsystem rather than to MVS.

command recognition character (CRC). A character that permits a z/OS console operator or an IMS subsystem user to route DB2 commands to specific DB2 subsystems.

command scope. The scope of command operation in a data sharing group. If a command has *member scope*, the command displays information only from the one member or affects only non-shared resources that are owned locally by that member. If a command has *group scope*, the command displays information from all members, affects non-shared resources that are owned locally by all members, displays information on sharable resources, or affects sharable resources.

commit. The operation that ends a unit of work by releasing locks so that the database changes that are made by that unit of work can be perceived by other processes.

commit point. A point in time when data is considered consistent.

committed phase. The second phase of the multisite update process that requests all participants to commit the effects of the logical unit of work.

common service area (CSA). In z/OS, a part of the common area that contains data areas that are addressable by all address spaces.

communications database (CDB). A set of tables in the DB2 catalog that are used to establish conversations with remote database management systems.

comparison operator. A token (such as =, >, or <) that is used to specify a relationship between two values.

composite key. An ordered set of key columns of the same table.

compression dictionary. The dictionary that controls the process of compression and decompression. This dictionary is created from the data in the table space or table space partition.

concurrency. The shared use of resources by more than one application process at the same time.

conditional restart. A DB2 restart that is directed by a user-defined conditional restart control record (CRCR).

connection. In SNA, the existence of a communication path between two partner LUs that allows information

to be exchanged (for example, two DB2 subsystems that are connected and communicating by way of a conversation).

connection context. In SQLJ, a Java object that represents a connection to a data source.

connection declaration clause. In SQLJ, a statement that declares a connection to a data source.

connection handle. The data object containing information that is associated with a connection that DB2 ODBC manages. This includes general status information, transaction status, and diagnostic information.

connection ID. An identifier that is supplied by the attachment facility and that is associated with a specific address space connection.

consistency token. A timestamp that is used to generate the version identifier for an application. See also *version*.

constant. A language element that specifies an unchanging value. Constants are classified as string constants or numeric constants. Contrast with *variable*.

constraint. A rule that limits the values that can be inserted, deleted, or updated in a table. See *referential constraint*, *check constraint*, and *unique constraint*.

context. The application's logical connection to the data source and associated internal DB2 ODBC connection information that allows the application to direct its operations to a data source. A DB2 ODBC context represents a DB2 thread.

contracting conversion. A process that occurs when the length of a converted string is smaller than that of the source string. For example, this process occurs when an EBCDIC mixed-data string that contains DBCS characters is converted to ASCII mixed data; the converted string is shorter because of the removal of the shift codes.

control interval (CI). A fixed-length area or disk in which VSAM stores records and creates distributed free space. Also, in a key-sequenced data set or file, the set of records that an entry in the sequence-set index record points to. The control interval is the unit of information that VSAM transmits to or from disk. A control interval always includes an integral number of physical records.

control interval definition field (CIDF). In VSAM, a field that is located in the 4 bytes at the end of each control interval; it describes the free space, if any, in the control interval.

conversation. Communication, which is based on LU 6.2 or Advanced Program-to-Program Communication (APPC), between an application and a remote transaction program over an SNA logical unit-to-logical unit (LU-LU) session that allows communication while processing a transaction.

coordinator. The system component that coordinates the commit or rollback of a unit of work that includes work that is done on one or more other systems.

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.

correlated columns. A relationship between the value of one column and the value of another column.

correlated subquery. A subquery (part of a WHERE or HAVING clause) that is applied to a row or group of rows of a table or view that is named in an outer subselect statement.

correlation ID. An identifier that is associated with a specific thread. In TSO, it is either an authorization ID or the job name.

correlation name. An identifier that designates a table, a view, or individual rows of a table or view within a single SQL statement. It can be defined in any FROM clause or in the first clause of an UPDATE or DELETE statement.

cost category. A category into which DB2 places cost estimates for SQL statements at the time the statement is bound. A cost estimate can be placed in either of the following cost categories:

- A: Indicates that DB2 had enough information to make a cost estimate without using default values.
- B: Indicates that some condition exists for which DB2 was forced to use default values for its estimate.

The cost category is externalized in the COST_CATEGORY column of the DSN_STATEMNT_TABLE when a statement is explained.

coupling facility. A special PR/SM™ LPAR logical partition that runs the coupling facility control program and provides high-speed caching, list processing, and locking functions in a Parallel Sysplex®.

coupling facility resource management. A component of z/OS that provides the services to manage coupling facility resources in a Parallel Sysplex. This management includes the enforcement of CFRM policies to ensure that the coupling facility and structure requirements are satisfied.

CP. Central processor.

CPC. Central processor complex.

C++ member. A data object or function in a structure, union, or class.

C++ member function. An operator or function that is declared as a member of a class. A member function has access to the private and protected data members and to the member functions of objects in its class. Member functions are also called methods.

C++ object. (1) A region of storage. An object is created when a variable is defined or a new function is invoked. (2) An instance of a class.

CRC. Command recognition character.

CRCR. Conditional restart control record. See also *conditional restart*.

create link pack area (CLPA). An option that is used during IPL to initialize the link pack pageable area.

created temporary table. A table that holds temporary data and is defined with the SQL statement CREATE GLOBAL TEMPORARY TABLE. Information about created temporary tables is stored in the DB2 catalog, so this kind of table is persistent and can be shared across application processes. Contrast with *declared temporary table*. See also *temporary table*.

cross-memory linkage. A method for invoking a program in a different address space. The invocation is synchronous with respect to the caller.

cross-system coupling facility (XCF). A component of z/OS that provides functions to support cooperation between authorized programs that run within a Sysplex.

cross-system extended services (XES). A set of z/OS services that allow multiple instances of an application or subsystem, running on different systems in a Sysplex environment, to implement high-performance, high-availability data sharing by using a coupling facility.

CS. Cursor stability.

CSA. Common service area.

CT. Cursor table.

current data. Data within a host structure that is current with (identical to) the data within the base table.

current SQL ID. An ID that, at a single point in time, holds the privileges that are exercised when certain dynamic SQL statements run. The current SQL ID can be a primary authorization ID or a secondary authorization ID.

current status rebuild. The second phase of restart processing during which the status of the subsystem is reconstructed from information on the log.

cursor. A named control structure that an application program uses to point to a single row or multiple rows within some ordered set of rows of a result table. A cursor can be used to retrieve, update, or delete rows from a result table.

cursor sensitivity. The degree to which database updates are visible to the subsequent FETCH statements in a cursor. A cursor can be sensitive to changes that are made with positioned update and delete statements specifying the name of that cursor. A cursor can also be sensitive to changes that are made with searched update or delete statements, or with cursors other than this cursor. These changes can be made by this application process or by another application process.

cursor stability (CS). The isolation level that provides maximum concurrency without the ability to read uncommitted data. With cursor stability, a unit of work holds locks only on its uncommitted changes and on the current row of each of its cursors.

cursor table (CT). The copy of the skeleton cursor table that is used by an executing application process.

cycle. A set of tables that can be ordered so that each table is a descendent of the one before it, and the first table is a descendent of the last table. A self-referencing table is a cycle with a single member.

D

DAD. See *Document access definition.*

disk. A direct-access storage device that records data magnetically.

database. A collection of tables, or a collection of table spaces and index spaces.

database access thread. A thread that accesses data at the local subsystem on behalf of a remote subsystem.

database administrator (DBA). An individual who is responsible for designing, developing, operating, safeguarding, maintaining, and using a database.

database alias. The name of the target server if different from the location name. The database alias name is used to provide the name of the database server as it is known to the network. When a database alias name is defined, the location name is used by the application to reference the server, but the database alias name is used to identify the database server to be accessed. Any fully qualified object names within any SQL statements are not modified and are sent unchanged to the database server.

database descriptor (DBD). An internal representation of a DB2 database definition, which reflects the data definition that is in the DB2 catalog. The objects that are defined in a database descriptor are table spaces, tables, indexes, index spaces, relationships, check constraints, and triggers. A DBD also contains information about accessing tables in the database.

database exception status. An indication that something is wrong with a database. All members of a data sharing group must know and share the exception status of databases.

database identifier (DBID). An internal identifier of the database.

database management system (DBMS). A software system that controls the creation, organization, and modification of a database and the access to the data that is stored within it.

database request module (DBRM). A data set member that is created by the DB2 precompiler and that contains information about SQL statements. DBRMs are used in the bind process.

database server. The target of a request from a local application or an intermediate database server. In the DB2 environment, the database server function is provided by the distributed data facility to access DB2 data from local applications, or from a remote database server that acts as an intermediate database server.

data currency. The state in which data that is retrieved into a host variable in your program is a copy of data in the base table.

data definition name (ddname). The name of a data definition (DD) statement that corresponds to a data control block containing the same name.

data dictionary. A repository of information about an organization's application programs, databases, logical data models, users, and authorizations. A data dictionary can be manual or automated.

data-driven business rules. Constraints on particular data values that exist as a result of requirements of the business.

Data Language/I (DL/I). The IMS data manipulation language; a common high-level interface between a user application and IMS.

data mart. A small data warehouse that applies to a single department or team. See also *data warehouse*.

data mining. The process of collecting critical business information from a data warehouse, correlating it, and uncovering associations, patterns, and trends.

data partition. A VSAM data set that is contained within a partitioned table space.

data-partitioned secondary index (DPSI). A secondary index that is partitioned. The index is partitioned according to the underlying data.

data sharing. The ability of two or more DB2 subsystems to directly access and change a single set of data.

data sharing group. A collection of one or more DB2 subsystems that directly access and change the same data while maintaining data integrity.

data sharing member. A DB2 subsystem that is assigned by XCF services to a data sharing group.

data source. A local or remote relational or non-relational data manager that is capable of supporting data access via an ODBC driver that supports the ODBC APIs. In the case of DB2 UDB for z/OS, the data sources are always relational database managers.

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.

data type. An attribute of columns, literals, host variables, special registers, and the results of functions and expressions.

data warehouse. A system that provides critical business information to an organization. The data warehouse system cleanses the data for accuracy and currency, and then presents the data to decision makers so that they can interpret and use it effectively and efficiently.

date. A three-part value that designates a day, month, and year.

date duration. A decimal integer that represents a number of years, months, and days.

datetime value. A value of the data type DATE, TIME, or TIMESTAMP.

DBA. Database administrator.

DBCLOB. Double-byte character large object.

DBCS. Double-byte character set.

DBD. Database descriptor.

DBID. Database identifier.

DBMS. Database management system.

DBRM. Database request module.

DB2 catalog. Tables that are maintained by DB2 and contain descriptions of DB2 objects, such as tables, views, and indexes.

DB2 command. An instruction to the DB2 subsystem that a user enters to start or stop DB2, to display information on current users, to start or stop databases, to display information on the status of databases, and so on.

DB2 for VSE & VM. The IBM DB2 relational database management system for the VSE and VM operating systems.

DB2I. DB2 Interactive.

DB2 Interactive (DB2I). The DB2 facility that provides for the execution of SQL statements, DB2 (operator) commands, programmer commands, and utility invocation.

DB2I Kanji Feature. The tape that contains the panels and jobs that allow a site to display DB2I panels in Kanji.

DB2 PM. DB2 Performance Monitor.

DB2 thread. The DB2 structure that describes an application's connection, traces its progress, processes resource functions, and delimits its accessibility to DB2 resources and services.

DCLGEN. Declarations generator.

DDF. Distributed data facility.

ddname. Data definition name.

deadlock. Unresolvable contention for the use of a resource, such as a table or an index.

declarations generator (DCLGEN). A subcomponent of DB2 that generates SQL table declarations and COBOL, C, or PL/I data structure declarations that conform to the table. The declarations are generated from DB2 system catalog information. DCLGEN is also a DSN subcommand.

declared temporary table. A table that holds temporary data and is defined with the SQL statement DECLARE GLOBAL TEMPORARY TABLE. Information about declared temporary tables is not stored in the DB2 catalog, so this kind of table is not persistent and can be used only by the application process that issued the DECLARE statement. Contrast with *created temporary table*. See also *temporary table*.

default value. A predetermined value, attribute, or option that is assumed when no other is explicitly specified.

deferred embedded SQL. SQL statements that are neither fully static nor fully dynamic. Like static statements, they are embedded within an application, but like dynamic statements, they are prepared during the execution of the application.

deferred write. The process of asynchronously writing changed data pages to disk.

degree of parallelism. The number of concurrently executed operations that are initiated to process a query.

delete-connected. A table that is a dependent of table P or a dependent of a table to which delete operations from table P cascade.

delete hole. The location on which a cursor is positioned when a row in a result table is refetched and the row no longer exists on the base table, because another cursor deleted the row between the time the cursor first included the row in the result table and the time the cursor tried to refetch it.

delete rule. The rule that tells DB2 what to do to a dependent row when a parent row is deleted. For each relationship, the rule might be CASCADE, RESTRICT, SET NULL, or NO ACTION.

delete trigger. A trigger that is defined with the triggering SQL operation DELETE.

delimited identifier. A sequence of characters that are enclosed within double quotation marks ("). The sequence must consist of a letter followed by zero or more characters, each of which is a letter, digit, or the underscore character (_).

delimiter token. A string constant, a delimited identifier, an operator symbol, or any of the special characters that are shown in DB2 syntax diagrams.

denormalization. A key step in the task of building a physical relational database design. Denormalization is the intentional duplication of columns in multiple tables, and the consequence is increased data redundancy. Denormalization is sometimes necessary to minimize performance problems. Contrast with *normalization*.

dependent. An object (row, table, or table space) that has at least one parent. The object is also said to be a dependent (row, table, or table space) of its parent. See also *parent row*, *parent table*, *parent table space*.

dependent row. A row that contains a foreign key that matches the value of a primary key in the parent row.

dependent table. A table that is a dependent in at least one referential constraint.

DES-based authenticator. An authenticator that is generated using the DES algorithm.

descendent. An object that is a dependent of an object or is the dependent of a descendent of an object.

descendent row. A row that is dependent on another row, or a row that is a descendent of a dependent row.

descendent table. A table that is a dependent of another table, or a table that is a descendent of a dependent table.

deterministic function. A user-defined function whose result is dependent on the values of the input arguments. That is, successive invocations with the same input values produce the same answer. Sometimes referred to as a *not-variant* function. Contrast this with an *nondeterministic function* (sometimes called a *variant function*), which might not always produce the same result for the same inputs.

DFP. Data Facility Product (in z/OS).

DFSMS. Data Facility Storage Management Subsystem (in z/OS). Also called *Storage Management Subsystem (SMS)*.

| **DFSMSdss.** The data set services (dss) component of
| DFSMS (in z/OS).

| **DFSMShsm™.** The hierarchical storage manager (hsm)
| component of DFSMS (in z/OS).

dimension. A data category such as time, products, or markets. The elements of a dimension are referred to as members. Dimensions offer a very concise, intuitive way of organizing and selecting data for retrieval, exploration, and analysis. See also *dimension table*.

dimension table. The representation of a dimension in a star schema. Each row in a dimension table represents all of the attributes for a particular member of the dimension. See also *dimension*, *star schema*, and *star join*.

directory. The DB2 system database that contains internal objects such as database descriptors and skeleton cursor tables.

distinct predicate. In SQL, a predicate that ensures
that two row values are not equal, and that both row
values are not null.

distinct type. A user-defined data type that is internally represented as an existing type (its source type), but is considered to be a separate and incompatible type for semantic purposes.

distributed data. Data that resides on a DBMS other than the local system.

distributed data facility (DDF). A set of DB2 components through which DB2 communicates with another relational database management system.

Distributed Relational Database Architecture™ (DRDA). A connection protocol for distributed relational database processing that is used by IBM's relational database products. DRDA includes protocols for communication between an application and a remote relational database management system, and for communication between relational database management systems. See also *DRDA access*.

DL/I. Data Language/I.

DNS. Domain name server.

| **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.

domain. The set of valid values for an attribute.

domain name. The name by which TCP/IP applications refer to a TCP/IP host within a TCP/IP network.

domain name server (DNS). A special TCP/IP network server that manages a distributed directory that is used to map TCP/IP host names to IP addresses.

double-byte character large object (DBCLOB). A sequence of bytes representing double-byte characters where the size of the values can be up to 2 GB. In general, DBCLOB values are used whenever a double-byte character string might exceed the limits of the VARGRAPHIC type.

double-byte character set (DBCS). A set of characters, which are used by national languages such as Japanese and Chinese, that have more symbols than can be represented by a single byte. Each character is 2 bytes in length. Contrast with *single-byte character set* and *multibyte character set*.

double-precision floating point number. A 64-bit approximate representation of a real number.

downstream. The set of nodes in the syncpoint tree that is connected to the local DBMS as a participant in the execution of a two-phase commit.

| **DPSI.** Data-partitioned secondary index.

drain. The act of acquiring a locked resource by quiescing access to that object.

drain lock. A lock on a claim class that prevents a claim from occurring.

DRDA. Distributed Relational Database Architecture.

DRDA access. An open method of accessing distributed data that you can use to can connect to another database server to execute packages that were previously bound at the server location. You use the SQL CONNECT statement or an SQL statement with a three-part name to identify the server. Contrast with *private protocol access*.

DSN. (1) The default DB2 subsystem name. (2) The name of the TSO command processor of DB2. (3) The first three characters of DB2 module and macro names.

duration. A number that represents an interval of time. See also *date duration*, *labeled duration*, and *time duration*.

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 dump. A dump that is issued during the execution of a program, usually under the control of that program.

dynamic SQL. SQL statements that are prepared and executed within an application program while the program is executing. In dynamic SQL, the SQL source is contained in host language variables rather than being coded into the application program. The SQL statement can change several times during the application program's execution.

dynamic statement cache pool. A cache, located above the 2-GB storage line, that holds dynamic statements.

E

EA-enabled table space. A table space or index space that is enabled for extended addressability and that contains individual partitions (or pieces, for LOB table spaces) that are greater than 4 GB.

EB. See *exabyte*.

EBCDIC. Extended binary coded decimal interchange code. An encoding scheme that is used to represent character data in the z/OS, VM, VSE, and iSeries® environments. Contrast with *ASCII* and *Unicode*.

e-business. The transformation of key business processes through the use of Internet technologies.

EDM pool. A pool of main storage that is used for database descriptors, application plans, authorization cache, application packages.

EID. Event identifier.

embedded SQL. SQL statements that are coded within an application program. See *static SQL*.

enclave. In Language Environment® , an independent collection of routines, one of which is designated as the main routine. An enclave is similar to a program or run unit.

encoding scheme. A set of rules to represent character data (ASCII, EBCDIC, or Unicode).

entity. A significant object of interest to an organization.

enumerated list. A set of DB2 objects that are defined with a LISTDEF utility control statement in which pattern-matching characters (*, %, _ or ?) are not used.

environment. A collection of names of logical and physical resources that are used to support the performance of a function.

environment handle. In DB2 ODBC, the data object that contains global information regarding the state of the application. An environment handle must be allocated before a connection handle can be allocated. Only one environment handle can be allocated per application.

EOM. End of memory.

EOT. End of task.

equijoin. A join operation in which the join-condition has the form *expression = expression*.

error page range. A range of pages that are considered to be physically damaged. DB2 does not allow users to access any pages that fall within this range.

escape character. The symbol that is used to enclose an SQL delimited identifier. The escape character is the double quotation mark ("), except in COBOL applications, where the user assigns the symbol, which is either a double quotation mark or an apostrophe (').

ESDS. Entry sequenced data set.

ESMT. External subsystem module table (in IMS).

EUR. IBM European Standards.

exabyte. For processor, real and virtual storage capacities and channel volume:
1 152 921 504 606 846 976 bytes or 2^{60}.

exception table. A table that holds rows that violate referential constraints or check constraints that the CHECK DATA utility finds.

exclusive lock. A lock that prevents concurrently executing application processes from reading or changing data. Contrast with *share lock*.

executable statement. An SQL statement that can be embedded in an application program, dynamically prepared and executed, or issued interactively.

execution context. In SQLJ, a Java object that can be used to control the execution of SQL statements.

exit routine. A user-written (or IBM-provided default) program that receives control from DB2 to perform specific functions. Exit routines run as extensions of DB2.

expanding conversion. A process that occurs when the length of a converted string is greater than that of the source string. For example, this process occurs when an ASCII mixed-data string that contains DBCS characters is converted to an EBCDIC mixed-data string; the converted string is longer because of the addition of shift codes.

explicit hierarchical locking. Locking that is used to make the parent-child relationship between resources known to IRLM. This kind of locking avoids global locking overhead when no inter-DB2 interest exists on a resource.

exposed name. A correlation name or a table or view name for which a correlation name is not specified. Names that are specified in a FROM clause are exposed or non-exposed.

expression. An operand or a collection of operators and operands that yields a single value.

extended recovery facility (XRF). A facility that minimizes the effect of failures in z/OS, VTAM , the host processor, or high-availability applications during sessions between high-availability applications and designated terminals. This facility provides an alternative subsystem to take over sessions from the failing subsystem.

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.

external function. A function for which the body is written in a programming language that takes scalar argument values and produces a scalar result for each invocation. Contrast with *sourced function*, *built-in function*, and *SQL function*.

external procedure. A user-written application program that can be invoked with the SQL CALL statement, which is written in a programming language. Contrast with *SQL procedure*.

external routine. A user-defined function or stored procedure that is based on code that is written in an external programming language.

external subsystem module table (ESMT). In IMS, the table that specifies which attachment modules must be loaded.

F

failed member state. A state of a member of a data sharing group. When a member fails, the XCF permanently records the failed member state. This state usually means that the member's task, address space, or z/OS system terminated before the state changed from active to quiesced.

fallback. The process of returning to a previous release of DB2 after attempting or completing migration to a current release.

false global lock contention. A contention indication from the coupling facility when multiple lock names are hashed to the same indicator and when no real contention exists.

fan set. A direct physical access path to data, which is provided by an index, hash, or link; a fan set is the means by which the data manager supports the ordering of data.

federated database. The combination of a DB2 Universal Database server (in Linux, UNIX, and Windows environments) and multiple data sources to which the server sends queries. In a federated database system, a client application can use a single SQL statement to join data that is distributed across multiple database management systems and can view the data as if it were local.

fetch orientation. The specification of the desired placement of the cursor as part of a FETCH statement (for example, BEFORE, AFTER, NEXT, PRIOR, CURRENT, FIRST, LAST, ABSOLUTE, and RELATIVE).

field procedure. A user-written exit routine that is designed to receive a single value and transform (encode or decode) it in any way the user can specify.

filter factor. A number between zero and one that estimates the proportion of rows in a table for which a predicate is true.

fixed-length string. A character or graphic string whose length is specified and cannot be changed. Contrast with *varying-length string*.

FlashCopy. A function on the IBM Enterprise Storage Server® that can create a point-in-time copy of data while an application is running.

foreign key. A column or set of columns in a dependent table of a constraint relationship. The key must have the same number of columns, with the same descriptions, as the primary key of the parent table.

Each foreign key value must either match a parent key value in the related parent table or be null.

forest. An ordered set of subtrees of XML nodes.

forget. In a two-phase commit operation, (1) the vote that is sent to the prepare phase when the participant has not modified any data. The forget vote allows a participant to release locks and forget about the logical unit of work. This is also referred to as the read-only vote. (2) The response to the *committed* request in the second phase of the operation.

forward log recovery. The third phase of restart processing during which DB2 processes the log in a forward direction to apply all REDO log records.

free space. The total amount of unused space in a page; that is, the space that is not used to store records or control information is free space.

full outer join. The result of a join operation that includes the matched rows of both tables that are being joined and preserves the unmatched rows of both tables. See also *join*.

fullselect. A subselect, a values-clause, or a number of both that are combined by set operators. *Fullselect* specifies a result table. If UNION is not used, the result of the fullselect is the result of the specified subselect.

fully escaped mapping. A mapping from an SQL identifier to an XML name when the SQL identifier is a column name.

function. A mapping, which is embodied as a program (the function body) that is invocable by means of zero or more input values (arguments) to a single value (the result). See also *aggregate function* and *scalar function*.

Functions can be user-defined, built-in, or generated by DB2. (See also *built-in function*, *cast function*, *external function*, *sourced function*, *SQL function*, and *user-defined function*.)

function definer. The authorization ID of the owner of the schema of the function that is specified in the CREATE FUNCTION statement.

function implementer. The authorization ID of the owner of the function program and function package.

function package. A package that results from binding the DBRM for a function program.

function package owner. The authorization ID of the user who binds the function program's DBRM into a function package.

function resolution. The process, internal to the DBMS, by which a function invocation is bound to a particular function instance. This process uses the function name, the data types of the arguments, and a list of the applicable schema names (called the *SQL path*) to make the selection. This process is sometimes called *function selection*.

function selection. See *function resolution*.

function signature. The logical concatenation of a fully qualified function name with the data types of all of its parameters.

G

GB. Gigabyte (1 073 741 824 bytes).

GBP. Group buffer pool.

GBP-dependent. The status of a page set or page set partition that is dependent on the group buffer pool. Either read/write interest is active among DB2 subsystems for this page set, or the page set has changed pages in the group buffer pool that have not yet been cast out to disk.

generalized trace facility (GTF). A z/OS service program that records significant system events such as I/O interrupts, SVC interrupts, program interrupts, or external interrupts.

generic resource name. A name that VTAM uses to represent several application programs that provide the same function in order to handle session distribution and balancing in a Sysplex environment.

getpage. An operation in which DB2 accesses a data page.

global lock. A lock that provides concurrency control within and among DB2 subsystems. The scope of the lock is across all DB2 subsystems of a data sharing group.

global lock contention. Conflicts on locking requests between different DB2 members of a data sharing group when those members are trying to serialize shared resources.

governor. See *resource limit facility*.

graphic string. A sequence of DBCS characters.

gross lock. The *shared*, *update*, or *exclusive* mode locks on a table, partition, or table space.

group buffer pool (GBP). A coupling facility cache structure that is used by a data sharing group to cache data and to ensure that the data is consistent for all members.

group buffer pool duplexing. The ability to write data to two instances of a group buffer pool structure: a *primary group buffer pool* and a *secondary group buffer*

pool. z/OS publications refer to these instances as the "old" (for primary) and "new" (for secondary) structures.

group level. The release level of a data sharing group, which is established when the first member migrates to a new release.

group name. The z/OS XCF identifier for a data sharing group.

group restart. A restart of at least one member of a data sharing group after the loss of either locks or the shared communications area.

GTF. Generalized trace facility.

H

handle. In DB2 ODBC, a variable that refers to a data structure and associated resources. See also *statement handle*, *connection handle*, and *environment handle*.

help panel. A screen of information that presents tutorial text to assist a user at the workstation or terminal.

heuristic damage. The inconsistency in data between one or more participants that results when a heuristic decision to resolve an indoubt LUW at one or more participants differs from the decision that is recorded at the coordinator.

heuristic decision. A decision that forces indoubt resolution at a participant by means other than automatic resynchronization between coordinator and participant.

hole. A row of the result table that cannot be accessed because of a delete or an update that has been performed on the row. See also *delete hole* and *update hole*.

home address space. The area of storage that z/OS currently recognizes as *dispatched*.

host. The set of programs and resources that are available on a given TCP/IP instance.

host expression. A Java variable or expression that is referenced by SQL clauses in an SQLJ application program.

host identifier. A name that is declared in the host program.

host language. A programming language in which you can embed SQL statements.

host program. An application program that is written in a host language and that contains embedded SQL statements.

host structure. In an application program, a structure that is referenced by embedded SQL statements.

host variable. In an application program, an application variable that is referenced by embedded SQL statements.

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.

HSM. Hierarchical storage manager.

HTML. Hypertext Markup Language, a standard method for presenting Web data to users.

HTTP. Hypertext Transfer Protocol, a communication protocol that the Web uses.

I

ICF. Integrated catalog facility.

IDCAMS. An IBM program that is used to process access method services commands. It can be invoked as a job or jobstep, from a TSO terminal, or from within a user's application program.

IDCAMS LISTCAT. A facility for obtaining information that is contained in the access method services catalog.

identify. A request that an attachment service program in an address space that is separate from DB2 issues thorough the z/OS subsystem interface to inform DB2 of its existence and to initiate the process of becoming connected to DB2.

identity column. A column that provides a way for DB2 to automatically generate a numeric value for each row. The generated values are unique if cycling is not used. Identity columns are defined with the AS IDENTITY clause. Uniqueness of values can be ensured by defining a unique index that contains only the identity column. A table can have no more than one identity column.

IFCID. Instrumentation facility component identifier.

IFI. Instrumentation facility interface.

IFI call. An invocation of the instrumentation facility interface (IFI) by means of one of its defined functions.

IFP. IMS Fast Path.

image copy. An exact reproduction of all or part of a table space. DB2 provides utility programs to make full image copies (to copy the entire table space) or incremental image copies (to copy only those pages that have been modified since the last image copy).

implied forget. In the presumed-abort protocol, an implied response of *forget* to the second-phase *committed* request from the coordinator. The response is implied when the participant responds to any subsequent request from the coordinator.

IMS. Information Management System.

IMS attachment facility. A DB2 subcomponent that uses z/OS subsystem interface (SSI) protocols and cross-memory linkage to process requests from IMS to DB2 and to coordinate resource commitment.

IMS DB. Information Management System Database.

IMS TM. Information Management System Transaction Manager.

in-abort. A status of a unit of recovery. If DB2 fails after a unit of recovery begins to be rolled back, but before the process is completed, DB2 continues to back out the changes during restart.

in-commit. A status of a unit of recovery. If DB2 fails after beginning its phase 2 commit processing, it "knows," when restarted, that changes made to data are consistent. Such units of recovery are termed *in-commit*.

independent. An object (row, table, or table space) that is neither a parent nor a dependent of another object.

index. A set of pointers that are logically ordered by the values of a key. Indexes can provide faster access to data and can enforce uniqueness on the rows in a table.

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.

index key. The set of columns in a table that is used to determine the order of index entries.

index partition. A VSAM data set that is contained within a partitioning index space.

index space. A page set that is used to store the entries of one index.

indicator column. A 4-byte value that is stored in a base table in place of a LOB column.

indicator variable. A variable that is used to represent the null value in an application program. If the value for the selected column is null, a negative value is placed in the indicator variable.

indoubt. A status of a unit of recovery. If DB2 fails after it has finished its phase 1 commit processing and before it has started phase 2, only the commit coordinator knows if an individual unit of recovery is to be committed or rolled back. At emergency restart, if DB2 lacks the information it needs to make this decision, the status of the unit of recovery is *indoubt* until DB2 obtains this information from the coordinator. More than one unit of recovery can be indoubt at restart.

indoubt resolution. The process of resolving the status of an indoubt logical unit of work to either the committed or the rollback state.

inflight. A status of a unit of recovery. If DB2 fails before its unit of recovery completes phase 1 of the commit process, it merely backs out the updates of its unit of recovery at restart. These units of recovery are termed *inflight*.

inheritance. The passing downstream of class resources or attributes from a parent class in the class hierarchy to a child class.

initialization file. For DB2 ODBC applications, a file containing values that can be set to adjust the performance of the database manager.

inline copy. A copy that is produced by the LOAD or REORG utility. The data set that the inline copy produces is logically equivalent to a full image copy that is produced by running the COPY utility with read-only access (SHRLEVEL REFERENCE).

inner join. The result of a join operation that includes only the matched rows of both tables that are being joined. See also *join*.

inoperative package. A package that cannot be used because one or more user-defined functions or procedures that the package depends on were dropped. Such a package must be explicitly rebound. Contrast with *invalid package.*

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.

insert trigger. A trigger that is defined with the triggering SQL operation INSERT.

install. The process of preparing a DB2 subsystem to operate as a z/OS subsystem.

installation verification scenario. A sequence of operations that exercises the main DB2 functions and tests whether DB2 was correctly installed.

instrumentation facility component identifier (IFCID). A value that names and identifies a trace record of an event that can be traced. As a parameter on the START TRACE and MODIFY TRACE commands, it specifies that the corresponding event is to be traced.

instrumentation facility interface (IFI). A programming interface that enables programs to obtain online trace data about DB2, to submit DB2 commands, and to pass data to DB2.

Interactive System Productivity Facility (ISPF). An IBM licensed program that provides interactive dialog services in a z/OS environment.

inter-DB2 R/W interest. A property of data in a table space, index, or partition that has been opened by more than one member of a data sharing group and that has been opened for writing by at least one of those members.

intermediate database server. The target of a request from a local application or a remote application requester that is forwarded to another database server. In the DB2 environment, the remote request is forwarded transparently to another database server if the object that is referenced by a three-part name does not reference the local location.

internationalization. The support for an encoding scheme that is able to represent the code points of characters from many different geographies and languages. To support all geographies, the Unicode standard requires more than 1 byte to represent a single character. See also *Unicode*.

internal resource lock manager (IRLM). A z/OS subsystem that DB2 uses to control communication and database locking.

International Organization for Standardization. An international body charged with creating standards to facilitate the exchange of goods and services as well as cooperation in intellectual, scientific, technological, and economic activity.

invalid package. A package that depends on an object (other than a user-defined function) that is dropped. Such a package is implicitly rebound on invocation. Contrast with *inoperative package.*

invariant character set. (1) A character set, such as the syntactic character set, whose code point assignments do not change from code page to code page. (2) A minimum set of characters that is available as part of all character sets.

IP address. A 4-byte value that uniquely identifies a TCP/IP host.

IRLM. Internal resource lock manager.

ISO. International Organization for Standardization.

isolation level. The degree to which a unit of work is isolated from the updating operations of other units of work. See also *cursor stability*, *read stability*, *repeatable read*, and *uncommitted read*.

ISPF. Interactive System Productivity Facility.

ISPF/PDF. Interactive System Productivity Facility/Program Development Facility.

iterator. In SQLJ, an object that contains the result set of a query. An iterator is equivalent to a cursor in other host languages.

iterator declaration clause. In SQLJ, a statement that generates an iterator declaration class. An iterator is an object of an iterator declaration class.

J

Japanese Industrial Standard. An encoding scheme that is used to process Japanese characters.

JAR. Java Archive.

Java Archive (JAR). A file format that is used for aggregating many files into a single file.

JCL. Job control language.

JDBC. A Sun Microsystems database application programming interface (API) for Java that allows programs to access database management systems by using callable SQL. JDBC does not require the use of an SQL preprocessor. In addition, JDBC provides an architecture that lets users add modules called *database drivers*, which link the application to their choice of database management systems at run time.

JES. Job Entry Subsystem.

JIS. Japanese Industrial Standard.

job control language (JCL). A control language that is used to identify a job to an operating system and to describe the job's requirements.

Job Entry Subsystem (JES). An IBM licensed program that receives jobs into the system and processes all output data that is produced by the jobs.

join. A relational operation that allows retrieval of data from two or more tables based on matching column values. See also *equijoin*, *full outer join*, *inner join*, *left outer join*, *outer join*, *and right outer join*.

K

KB. Kilobyte (1024 bytes).

Kerberos. A network authentication protocol that is designed to provide strong authentication for client/server applications by using secret-key cryptography.

Kerberos ticket. A transparent application mechanism that transmits the identity of an initiating principal to its target. A simple ticket contains the principal's

identity, a session key, a timestamp, and other information, which is sealed using the target's secret key.

key. A column or an ordered collection of columns that is identified in the description of a table, index, or referential constraint. The same column can be part of more than one key.

key-sequenced data set (KSDS). A VSAM file or data set whose records are loaded in key sequence and controlled by an index.

keyword. In SQL, a name that identifies an option that is used in an SQL statement.

KSDS. Key-sequenced data set.

L

labeled duration. A number that represents a duration of years, months, days, hours, minutes, seconds, or microseconds.

large object (LOB). A sequence of bytes representing bit data, single-byte characters, double-byte characters, or a mixture of single- and double-byte characters. A LOB can be up to 2 GB–1 byte in length. See also *BLOB*, *CLOB*, and *DBCLOB*.

last agent optimization. An optimized commit flow for either presumed-nothing or presumed-abort protocols in which the last agent, or final participant, becomes the commit coordinator. This flow saves at least one message.

latch. A DB2 internal mechanism for controlling concurrent events or the use of system resources.

LCID. Log control interval definition.

LDS. Linear data set.

leaf page. A page that contains pairs of keys and RIDs and that points to actual data. Contrast with *nonleaf page*.

left outer join. The result of a join operation that includes the matched rows of both tables that are being joined, and that preserves the unmatched rows of the first table. See also *join*.

limit key. The highest value of the index key for a partition.

linear data set (LDS). A VSAM data set that contains data but no control information. A linear data set can be accessed as a byte-addressable string in virtual storage.

linkage editor. A computer program for creating load modules from one or more object modules or load modules by resolving cross references among the modules and, if necessary, adjusting addresses.

link-edit. The action of creating a loadable computer program using a linkage editor.

list. A type of object, which DB2 utilities can process, that identifies multiple table spaces, multiple index spaces, or both. A list is defined with the LISTDEF utility control statement.

list structure. A coupling facility structure that lets data be shared and manipulated as elements of a queue.

LLE. Load list element.

L-lock. Logical lock.

load list element. A z/OS control block that controls the loading and deleting of a particular load module based on entry point names.

load module. A program unit that is suitable for loading into main storage for execution. The output of a linkage editor.

LOB. Large object.

LOB locator. A mechanism that allows an application program to manipulate a large object value in the database system. A LOB locator is a fullword integer value that represents a single LOB value. An application program retrieves a LOB locator into a host variable and can then apply SQL operations to the associated LOB value using the locator.

LOB lock. A lock on a LOB value.

LOB table space. A table space in an auxiliary table that contains all the data for a particular LOB column in the related base table.

local. A way of referring to any object that the local DB2 subsystem maintains. A *local table*, for example, is a table that is maintained by the local DB2 subsystem. Contrast with *remote*.

locale. The definition of a subset of a user's environment that combines a CCSID and characters that are defined for a specific language and country.

local lock. A lock that provides intra-DB2 concurrency control, but not inter-DB2 concurrency control; that is, its scope is a single DB2.

local subsystem. The unique relational DBMS to which the user or application program is directly connected (in the case of DB2, by one of the DB2 attachment facilities).

location. The unique name of a database server. An application uses the location name to access a DB2

database server. A database alias can be used to override the location name when accessing a remote server.

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.

lock. A means of controlling concurrent events or access to data. DB2 locking is performed by the IRLM.

lock duration. The interval over which a DB2 lock is held.

lock escalation. The promotion of a lock from a row, page, or LOB lock to a table space lock because the number of page locks that are concurrently held on a given resource exceeds a preset limit.

locking. The process by which the integrity of data is ensured. Locking prevents concurrent users from accessing inconsistent data.

lock mode. A representation for the type of access that concurrently running programs can have to a resource that a DB2 lock is holding.

lock object. The resource that is controlled by a DB2 lock.

lock promotion. The process of changing the size or mode of a DB2 lock to a higher, more restrictive level.

lock size. The amount of data that is controlled by a DB2 lock on table data; the value can be a row, a page, a LOB, a partition, a table, or a table space.

lock structure. A coupling facility data structure that is composed of a series of lock entries to support shared and exclusive locking for logical resources.

log. A collection of records that describe the events that occur during DB2 execution and that indicate their sequence. The information thus recorded is used for recovery in the event of a failure during DB2 execution.

log control interval definition. A suffix of the physical log record that tells how record segments are placed in the physical control interval.

logical claim. A claim on a logical partition of a nonpartitioning index.

logical data modeling. The process of documenting the comprehensive business information requirements in an accurate and consistent format. Data modeling is the first task of designing a database.

logical drain. A drain on a logical partition of a nonpartitioning index.

logical index partition. The set of all keys that reference the same data partition.

logical lock (L-lock). The lock type that transactions use to control intra- and inter-DB2 data concurrency between transactions. Contrast with *physical lock (P-lock)*.

logically complete. A state in which the concurrent copy process is finished with the initialization of the target objects that are being copied. The target objects are available for update.

logical page list (LPL). A list of pages that are in error and that cannot be referenced by applications until the pages are recovered. The page is in *logical error* because the actual media (coupling facility or disk) might not contain any errors. Usually a connection to the media has been lost.

logical partition. A set of key or RID pairs in a nonpartitioning index that are associated with a particular partition.

logical recovery pending (LRECP). The state in which the data and the index keys that reference the data are inconsistent.

logical unit (LU). An access point through which an application program accesses the SNA network in order to communicate with another application program.

logical unit of work (LUW). The processing that a program performs between synchronization points.

logical unit of work identifier (LUWID). A name that uniquely identifies a thread within a network. This name consists of a fully-qualified LU network name, an LUW instance number, and an LUW sequence number.

log initialization. The first phase of restart processing during which DB2 attempts to locate the current end of the log.

log record header (LRH). A prefix, in every logical record, that contains control information.

log record sequence number (LRSN). A unique identifier for a log record that is associated with a data sharing member. DB2 uses the LRSN for recovery in the data sharing environment.

log truncation. A process by which an explicit starting RBA is established. This RBA is the point at which the next byte of log data is to be written.

LPL. Logical page list.

LRECP. Logical recovery pending.

LRH. Log record header.

LRSN. Log record sequence number.

LU. Logical unit.

LU name. Logical unit name, which is the name by which VTAM refers to a node in a network. Contrast with *location name*.

LUW. Logical unit of work.

LUWID. Logical unit of work identifier.

M

mapping table. A table that the REORG utility uses to map the associations of the RIDs of data records in the original copy and in the shadow copy. This table is created by the user.

mass delete. The deletion of all rows of a table.

master terminal. The IMS logical terminal that has complete control of IMS resources during online operations.

master terminal operator (MTO). See *master terminal.*

materialize. (1) The process of putting rows from a view or nested table expression into a work file for additional processing by a query.

(2) The placement of a LOB value into contiguous storage. Because LOB values can be very large, DB2 avoids materializing LOB data until doing so becomes absolutely necessary.

| **materialized query table.** A table that is used to
| contain information that is derived and can be
| summarized from one or more source tables.

MB. Megabyte (1 048 576 bytes).

MBCS. Multibyte character set. UTF-8 is an example of an MBCS. Characters in UTF-8 can range from 1 to 4 bytes in DB2.

member name. The z/OS XCF identifier for a particular DB2 subsystem in a data sharing group.

menu. A displayed list of available functions for selection by the operator. A menu is sometimes called a *menu panel*.

| **metalanguage.** A language that is used to create other
| specialized languages.

migration. The process of converting a subsystem with a previous release of DB2 to an updated or current release. In this process, you can acquire the functions of the updated or current release without losing the data that you created on the previous release.

mixed data string. A character string that can contain both single-byte and double-byte characters.

MLPA. Modified link pack area.

MODEENT. A VTAM macro instruction that associates a logon mode name with a set of parameters representing session protocols. A set of MODEENT macro instructions defines a logon mode table.

modeling database. A DB2 database that you create on your workstation that you use to model a DB2 UDB for z/OS subsystem, which can then be evaluated by the Index Advisor.

mode name. A VTAM name for the collection of physical and logical characteristics and attributes of a session.

modify locks. An L-lock or P-lock with a MODIFY attribute. A list of these active locks is kept at all times in the coupling facility lock structure. If the requesting DB2 subsystem fails, that DB2 subsystem's modify locks are converted to retained locks.

MPP. Message processing program (in IMS).

MTO. Master terminal operator.

multibyte character set (MBCS). A character set that represents single characters with more than a single byte. Contrast with *single-byte character set* and *double-byte character set*. See also *Unicode*.

multidimensional analysis. The process of assessing and evaluating an enterprise on more than one level.

Multiple Virtual Storage. An element of the z/OS operating system. This element is also called the Base Control Program (BCP).

multisite update. Distributed relational database processing in which data is updated in more than one location within a single unit of work.

multithreading. Multiple TCBs that are executing one copy of DB2 ODBC code concurrently (sharing a processor) or in parallel (on separate central processors).

must-complete. A state during DB2 processing in which the entire operation must be completed to maintain data integrity.

mutex. Pthread mutual exclusion; a lock. A Pthread mutex variable is used as a locking mechanism to allow serialization of critical sections of code by temporarily blocking the execution of all but one thread.

| **MVS.** See *Multiple Virtual Storage*.

N

negotiable lock. A lock whose mode can be downgraded, by agreement among contending users, to be compatible to all. A physical lock is an example of a negotiable lock.

nested table expression. A fullselect in a FROM clause (surrounded by parentheses).

network identifier (NID). The network ID that is assigned by IMS or CICS, or if the connection type is RRSAF, the RRS unit of recovery ID (URID).

NID. Network identifier.

nonleaf page. A page that contains keys and page numbers of other pages in the index (either leaf or nonleaf pages). Nonleaf pages never point to actual data.

nonpartitioned index. An index that is not physically partitioned. Both partitioning indexes and secondary indexes can be nonpartitioned.

nonscrollable cursor. A cursor that can be moved only in a forward direction. Nonscrollable cursors are sometimes called forward-only cursors or serial cursors.

normalization. A key step in the task of building a logical relational database design. Normalization helps you avoid redundancies and inconsistencies in your data. An entity is normalized if it meets a set of constraints for a particular normal form (first normal form, second normal form, and so on). Contrast with *denormalization*.

nondeterministic function. A user-defined function whose result is not solely dependent on the values of the input arguments. That is, successive invocations with the same argument values can produce a different answer. this type of function is sometimes called a *variant* function. Contrast this with a *deterministic function* (sometimes called a *not-variant function*), which always produces the same result for the same inputs.

not-variant function. See *deterministic function*.

NPSI. See *nonpartitioned secondary index.*

NRE. Network recovery element.

NUL. The null character ('\0'), which is represented by the value X'00'. In C, this character denotes the end of a string.

null. A special value that indicates the absence of information.

NULLIF. A scalar function that evaluates two passed expressions, returning either NULL if the arguments are equal or the value of the first argument if they are not.

null-terminated host variable. A varying-length host variable in which the end of the data is indicated by a null terminator.

null terminator. In C, the value that indicates the end of a string. For EBCDIC, ASCII, and Unicode UTF-8 strings, the null terminator is a single-byte value (X'00'). For Unicode UCS-2 (wide) strings, the null terminator is a double-byte value (X'0000').

O

OASN (origin application schedule number). In IMS, a 4-byte number that is assigned sequentially to each IMS schedule since the last cold start of IMS. The OASN is used as an identifier for a unit of work. In an 8-byte format, the first 4 bytes contain the schedule number and the last 4 bytes contain the number of IMS sync points (*commit points*) during the current schedule. The OASN is part of the NID for an IMS connection.

ODBC. Open Database Connectivity.

ODBC driver. A dynamically-linked library (DLL) that implements ODBC function calls and interacts with a data source.

OBID. Data object identifier.

Open Database Connectivity (ODBC). A Microsoft database application programming interface (API) for C that allows access to database management systems by using callable SQL. ODBC does not require the use of an SQL preprocessor. In addition, ODBC provides an architecture that lets users add modules called *database drivers*, which link the application to their choice of database management systems at run time. This means that applications no longer need to be directly linked to the modules of all the database management systems that are supported.

ordinary identifier. An uppercase letter followed by zero or more characters, each of which is an uppercase letter, a digit, or the underscore character. An ordinary identifier must not be a reserved word.

ordinary token. A numeric constant, an ordinary identifier, a host identifier, or a keyword.

originating task. In a parallel group, the primary agent that receives data from other execution units (referred to as *parallel tasks*) that are executing portions of the query in parallel.

OS/390. Operating System/390®.

outer join. The result of a join operation that includes the matched rows of both tables that are being joined and preserves some or all of the unmatched rows of the tables that are being joined. See also *join*.

overloaded function. A function name for which multiple function instances exist.

P

package. An object containing a set of SQL statements that have been statically bound and that is available for processing. A package is sometimes also called an *application package*.

package list. An ordered list of package names that may be used to extend an application plan.

package name. The name of an object that is created by a BIND PACKAGE or REBIND PACKAGE command. The object is a bound version of a database request module (DBRM). The name consists of a location name, a collection ID, a package ID, and a version ID.

page. A unit of storage within a table space (4 KB, 8 KB, 16 KB, or 32 KB) or index space (4 KB). In a table space, a page contains one or more rows of a table. In a LOB table space, a LOB value can span more than one page, but no more than one LOB value is stored on a page.

page set. Another way to refer to a table space or index space. Each page set consists of a collection of VSAM data sets.

page set recovery pending (PSRCP). A restrictive state of an index space. In this case, the entire page set must be recovered. Recovery of a logical part is prohibited.

panel. A predefined display image that defines the locations and characteristics of display fields on a display surface (for example, a *menu panel*).

parallel complex. A cluster of machines that work together to handle multiple transactions and applications.

parallel group. A set of consecutive operations that execute in parallel and that have the same number of parallel tasks.

parallel I/O processing. A form of I/O processing in which DB2 initiates multiple concurrent requests for a single user query and performs I/O processing concurrently (in *parallel*) on multiple data partitions.

parallelism assistant. In Sysplex query parallelism, a DB2 subsystem that helps to process parts of a parallel query that originates on another DB2 subsystem in the data sharing group.

parallelism coordinator. In Sysplex query parallelism, the DB2 subsystem from which the parallel query originates.

Parallel Sysplex. A set of z/OS systems that communicate and cooperate with each other through certain multisystem hardware components and software services to process customer workloads.

parallel task. The execution unit that is dynamically created to process a query in parallel. A parallel task is implemented by a z/OS service request block.

parameter marker. A question mark (?) that appears in a statement string of a dynamic SQL statement. The question mark can appear where a host variable could appear if the statement string were a static SQL statement.

parameter-name. An SQL identifier that designates a parameter in an SQL procedure or an SQL function.

parent key. A primary key or unique key in the parent table of a referential constraint. The values of a parent key determine the valid values of the foreign key in the referential constraint.

parent lock. For explicit hierarchical locking, a lock that is held on a resource that might have child locks that are lower in the hierarchy. A parent lock is usually the table space lock or the partition intent lock. See also *child lock*.

parent row. A row whose primary key value is the foreign key value of a dependent row.

parent table. A table whose primary key is referenced by the foreign key of a dependent table.

parent table space. A table space that contains a parent table. A table space containing a dependent of that table is a dependent table space.

participant. An entity other than the commit coordinator that takes part in the commit process. The term participant is synonymous with *agent* in SNA.

partition. A portion of a page set. Each partition corresponds to a single, independently extendable data set. Partitions can be extended to a maximum size of 1, 2, or 4 GB, depending on the number of partitions in the partitioned page set. All partitions of a given page set have the same maximum size.

partitioned data set (PDS). A data set in disk storage that is divided into partitions, which are called members. Each partition can contain a program, part of a program, or data. The term partitioned data set is synonymous with program library.

partitioned index. An index that is physically partitioned. Both partitioning indexes and secondary indexes can be partitioned.

partitioned page set. A partitioned table space or an index space. Header pages, space map pages, data pages, and index pages reference data only within the scope of the partition.

partitioned table space. A table space that is subdivided into parts (based on index key range), each of which can be processed independently by utilities.

partitioning index. An index in which the leftmost columns are the partitioning columns of the table. The index can be partitioned or nonpartitioned.

partition pruning. The removal from consideration of inapplicable partitions through setting up predicates in a query on a partitioned table to access only certain partitions to satisfy the query.

partner logical unit. An access point in the SNA network that is connected to the local DB2 subsystem by way of a VTAM conversation.

path. See *SQL path*.

PCT. Program control table (in CICS).

PDS. Partitioned data set.

piece. A data set of a nonpartitioned page set.

physical claim. A claim on an entire nonpartitioning index.

physical consistency. The state of a page that is not in a partially changed state.

physical drain. A drain on an entire nonpartitioning index.

physical lock (P-lock). A type of lock that DB2 acquires to provide consistency of data that is cached in different DB2 subsystems. Physical locks are used only in data sharing environments. Contrast with *logical lock (L-lock)*.

physical lock contention. Conflicting states of the requesters for a physical lock. See also *negotiable lock*.

physically complete. The state in which the concurrent copy process is completed and the output data set has been created.

plan. See *application plan*.

plan allocation. The process of allocating DB2 resources to a plan in preparation for execution.

plan member. The bound copy of a DBRM that is identified in the member clause.

plan name. The name of an application plan.

plan segmentation. The dividing of each plan into sections. When a section is needed, it is independently brought into the EDM pool.

P-lock. Physical lock.

PLT. Program list table (in CICS).

point of consistency. A time when all recoverable data that an application accesses is consistent with other data. The term point of consistency is synonymous with *sync point* or *commit point*.

policy. See *CFRM policy*.

Portable Operating System Interface (POSIX). The IEEE operating system interface standard, which defines the Pthread standard of threading. See also *Pthread*.

POSIX. Portable Operating System Interface.

postponed abort UR. A unit of recovery that was inflight or in-abort, was interrupted by system failure or cancellation, and did not complete backout during restart.

PPT. (1) Processing program table (in CICS). (2) Program properties table (in z/OS).

precision. In SQL, the total number of digits in a decimal number (called the *size* in the C language). In the C language, the number of digits to the right of the decimal point (called the *scale* in SQL). The DB2 library uses the SQL terms.

precompilation. A processing of application programs containing SQL statements that takes place before compilation. SQL statements are replaced with statements that are recognized by the host language compiler. Output from this precompilation includes source code that can be submitted to the compiler and the database request module (DBRM) that is input to the bind process.

predicate. An element of a search condition that expresses or implies a comparison operation.

prefix. A code at the beginning of a message or record.

preformat. The process of preparing a VSAM ESDS for DB2 use, by writing specific data patterns.

prepare. The first phase of a two-phase commit process in which all participants are requested to prepare for commit.

prepared SQL statement. A named object that is the executable form of an SQL statement that has been processed by the PREPARE statement.

presumed-abort. An optimization of the presumed-nothing two-phase commit protocol that reduces the number of recovery log records, the duration of state maintenance, and the number of messages between coordinator and participant. The optimization also modifies the indoubt resolution responsibility.

presumed-nothing. The standard two-phase commit protocol that defines coordinator and participant responsibilities, relative to logical unit of work states, recovery logging, and indoubt resolution.

primary authorization ID. The authorization ID that is used to identify the application process to DB2.

primary group buffer pool. For a duplexed group buffer pool, the structure that is used to maintain the coherency of cached data. This structure is used for page registration and cross-invalidation. The z/OS equivalent is *old* structure. Compare with *secondary group buffer pool*.

primary index. An index that enforces the uniqueness of a primary key.

primary key. In a relational database, a unique, nonnull key that is part of the definition of a table. A table cannot be defined as a parent unless it has a unique key or primary key.

principal. An entity that can communicate securely with another entity. In Kerberos, principals are represented as entries in the Kerberos registry database and include users, servers, computers, and others.

principal name. The name by which a principal is known to the DCE security services.

private connection. A communications connection that is specific to DB2.

private protocol access. A method of accessing distributed data by which you can direct a query to another DB2 system. Contrast with *DRDA access*.

private protocol connection. A DB2 private connection of the application process. See also *private connection*.

privilege. The capability of performing a specific function, sometimes on a specific object. The types of privileges are:

explicit privileges, which have names and are held as the result of SQL GRANT and REVOKE statements. For example, the SELECT privilege.
implicit privileges, which accompany the ownership of an object, such as the privilege to drop a synonym that one owns, or the holding of an authority, such as the privilege of SYSADM authority to terminate any utility job.

privilege set. For the installation SYSADM ID, the set of all possible privileges. For any other authorization ID, the set of all privileges that are recorded for that ID in the DB2 catalog.

process. In DB2, the unit to which DB2 allocates resources and locks. Sometimes called an *application process*, a process involves the execution of one or more programs. The execution of an SQL statement is always associated with some process. The means of initiating and terminating a process are dependent on the environment.

program. A single, compilable collection of executable statements in a programming language.

program temporary fix (PTF). A solution or bypass of a problem that is diagnosed as a result of a defect in a current unaltered release of a licensed program. An authorized program analysis report (APAR) fix is corrective service for an existing problem. A PTF is preventive service for problems that might be encountered by other users of the product. A PTF is *temporary*, because a permanent fix is usually not incorporated into the product until its next release.

protected conversation. A VTAM conversation that supports two-phase commit flows.

PSRCP. Page set recovery pending.

PTF. Program temporary fix.

Pthread. The POSIX threading standard model for splitting an application into subtasks. The Pthread standard includes functions for creating threads, terminating threads, synchronizing threads through locking, and other thread control facilities.

Q

QMF™. Query Management Facility.

QSAM. Queued sequential access method.

query. A component of certain SQL statements that specifies a result table.

query block. The part of a query that is represented by one of the FROM clauses. Each FROM clause can have multiple query blocks, depending on DB2's internal processing of the query.

query CP parallelism. Parallel execution of a single query, which is accomplished by using multiple tasks. See also *Sysplex query parallelism*.

query I/O parallelism. Parallel access of data, which is accomplished by triggering multiple I/O requests within a single query.

queued sequential access method (QSAM). An extended version of the basic sequential access method (BSAM). When this method is used, a queue of data blocks is formed. Input data blocks await processing, and output data blocks await transfer to auxiliary storage or to an output device.

quiesce point. A point at which data is consistent as a result of running the DB2 QUIESCE utility.

quiesced member state. A state of a member of a data sharing group. An active member becomes quiesced when a STOP DB2 command takes effect without a failure. If the member's task, address space, or z/OS system fails before the command takes effect, the member state is failed.

R

RACF. Resource Access Control Facility, which is a component of the z/OS Security Server.

RAMAC®. IBM family of enterprise disk storage system products.

RBA. Relative byte address.

RCT. Resource control table (in CICS attachment facility).

RDB. Relational database.

RDBMS. Relational database management system.

RDBNAM. Relational database name.

RDF. Record definition field.

read stability (RS). An isolation level that is similar to repeatable read but does not completely isolate an application process from all other concurrently executing application processes. Under level RS, an application that issues the same query more than once might read additional rows that were inserted and committed by a concurrently executing application process.

rebind. The creation of a new application plan for an application program that has been bound previously. If, for example, you have added an index for a table that your application accesses, you must rebind the application in order to take advantage of that index.

rebuild. The process of reallocating a coupling facility structure. For the shared communications area (SCA) and lock structure, the structure is repopulated; for the group buffer pool, changed pages are usually cast out to disk, and the new structure is populated only with changed pages that were not successfully cast out.

RECFM. Record format.

record. The storage representation of a row or other data.

record identifier (RID). A unique identifier that DB2 uses internally to identify a row of data in a table. Compare with *row ID*.

record identifier (RID) pool. An area of main storage that is used for sorting record identifiers during list-prefetch processing.

record length. The sum of the length of all the columns in a table, which is the length of the data as it is physically stored in the database. Records can be fixed length or varying length, depending on how the columns are defined. If all columns are fixed-length columns, the record is a fixed-length record. If one or more columns are varying-length columns, the record is a varying-length column.

Recoverable Resource Manager Services attachment facility (RRSAF). A DB2 subcomponent that uses Resource Recovery Services to coordinate resource commitment between DB2 and all other resource managers that also use RRS in a z/OS system.

recovery. The process of rebuilding databases after a system failure.

recovery log. A collection of records that describes the events that occur during DB2 execution and indicates their sequence. The recorded information is used for recovery in the event of a failure during DB2 execution.

recovery manager. (1) A subcomponent that supplies coordination services that control the interaction of DB2 resource managers during commit, abort, checkpoint, and restart processes. The recovery manager also supports the recovery mechanisms of other subsystems (for example, IMS) by acting as a participant in the other subsystem's process for protecting data that has reached a point of consistency. (2) A coordinator or a participant (or both), in the execution of a two-phase commit, that can access a recovery log that maintains the state of the logical unit of work and names the immediate upstream coordinator and downstream participants.

recovery pending (RECP). A condition that prevents SQL access to a table space that needs to be recovered.

recovery token. An identifier for an element that is used in recovery (for example, NID or URID).

RECP. Recovery pending.

redo. A state of a unit of recovery that indicates that changes are to be reapplied to the disk media to ensure data integrity.

reentrant. Executable code that can reside in storage as one shared copy for all threads. Reentrant code is not self-modifying and provides separate storage areas for each thread. Reentrancy is a compiler and operating system concept, and reentrancy alone is not enough to guarantee logically consistent results when multithreading. See also *threadsafe*.

referential constraint. The requirement that nonnull values of a designated foreign key are valid only if they equal values of the primary key of a designated table.

referential integrity. The state of a database in which all values of all foreign keys are valid. Maintaining referential integrity requires the enforcement of referential constraints on all operations that change the data in a table on which the referential constraints are defined.

referential structure. A set of tables and relationships that includes at least one table and, for every table in the set, all the relationships in which that table participates and all the tables to which it is related.

refresh age. The time duration between the current time and the time during which a materialized query table was last refreshed.

registry. See *registry database*.

registry database. A database of security information about principals, groups, organizations, accounts, and security policies.

relational database (RDB). A database that can be perceived as a set of tables and manipulated in accordance with the relational model of data.

relational database management system (RDBMS). A collection of hardware and software that organizes and provides access to a relational database.

relational database name (RDBNAM). A unique identifier for an RDBMS within a network. In DB2, this must be the value in the LOCATION column of table SYSIBM.LOCATIONS in the CDB. DB2 publications refer to the name of another RDBMS as a LOCATION value or a location name.

relationship. A defined connection between the rows of a table or the rows of two tables. A relationship is the internal representation of a referential constraint.

relative byte address (RBA). The offset of a data record or control interval from the beginning of the storage space that is allocated to the data set or file to which it belongs.

remigration. The process of returning to a current release of DB2 following a fallback to a previous release. This procedure constitutes another migration process.

remote. Any object that is maintained by a remote DB2 subsystem (that is, by a DB2 subsystem other than the local one). A *remote view*, for example, is a view that is maintained by a remote DB2 subsystem. Contrast with *local*.

remote attach request. A request by a remote location to attach to the local DB2 subsystem. Specifically, the request that is sent is an SNA Function Management Header 5.

remote subsystem. Any relational DBMS, except the *local subsystem*, with which the user or application can communicate. The subsystem need not be remote in any physical sense, and might even operate on the same processor under the same z/OS system.

reoptimization. The DB2 process of reconsidering the access path of an SQL statement at run time; during reoptimization, DB2 uses the values of host variables, parameter markers, or special registers.

REORG pending (REORP). A condition that restricts SQL access and most utility access to an object that must be reorganized.

REORP. REORG pending.

repeatable read (RR). The isolation level that provides maximum protection from other executing application programs. When an application program executes with repeatable read protection, rows that the program references cannot be changed by other programs until the program reaches a commit point.

repeating group. A situation in which an entity includes multiple attributes that are inherently the same. The presence of a repeating group violates the requirement of first normal form. In an entity that satisfies the requirement of first normal form, each attribute is independent and unique in its meaning and its name. See also *normalization*.

replay detection mechanism. A method that allows a principal to detect whether a request is a valid request from a source that can be trusted or whether an untrustworthy entity has captured information from a previous exchange and is replaying the information exchange to gain access to the principal.

request commit. The vote that is submitted to the prepare phase if the participant has modified data and is prepared to commit or roll back.

requester. The source of a request to access data at a remote server. In the DB2 environment, the requester function is provided by the distributed data facility.

resource. The object of a lock or claim, which could be a table space, an index space, a data partition, an index partition, or a logical partition.

resource allocation. The part of plan allocation that deals specifically with the database resources.

resource control table (RCT). A construct of the CICS attachment facility, created by site-provided macro parameters, that defines authorization and access attributes for transactions or transaction groups.

resource definition online. A CICS feature that you use to define CICS resources online without assembling tables.

resource limit facility (RLF). A portion of DB2 code that prevents dynamic manipulative SQL statements from exceeding specified time limits. The resource limit facility is sometimes called the governor.

resource limit specification table (RLST). A site-defined table that specifies the limits to be enforced by the resource limit facility.

resource manager. (1) A function that is responsible for managing a particular resource and that guarantees the consistency of all updates made to recoverable resources within a logical unit of work. The resource that is being managed can be physical (for example, disk or main storage) or logical (for example, a particular type of system service). (2) A participant, in the execution of a two-phase commit, that has recoverable resources that could have been modified. The resource manager has access to a recovery log so that it can commit or roll back the effects of the logical unit of work to the recoverable resources.

restart pending (RESTP). A restrictive state of a page set or partition that indicates that restart (backout) work needs to be performed on the object. All access to the page set or partition is denied except for access by the:
- RECOVER POSTPONED command
- Automatic online backout (which DB2 invokes after restart if the system parameter LBACKOUT=AUTO)

RESTP. Restart pending.

result set. The set of rows that a stored procedure returns to a client application.

result set locator. A 4-byte value that DB2 uses to uniquely identify a query result set that a stored procedure returns.

result table. The set of rows that are specified by a SELECT statement.

retained lock. A MODIFY lock that a DB2 subsystem was holding at the time of a subsystem failure. The lock is retained in the coupling facility lock structure across a DB2 failure.

RID. Record identifier.

RID pool. Record identifier pool.

right outer join. The result of a join operation that includes the matched rows of both tables that are being joined and preserves the unmatched rows of the second join operand. See also *join*.

RLF. Resource limit facility.

RLST. Resource limit specification table.

RMID. Resource manager identifier.

RO. Read-only access.

rollback. The process of restoring data that was changed by SQL statements to the state at its last commit point. All locks are freed. Contrast with *commit*.

root page. The index page that is at the highest level (or the beginning point) in an index.

routine. A term that refers to either a user-defined function or a stored procedure.

row. The horizontal component of a table. A row consists of a sequence of values, one for each column of the table.

ROWID. Row identifier.

row identifier (ROWID). A value that uniquely identifies a row. This value is stored with the row and never changes.

row lock. A lock on a single row of data.

| **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.

row trigger. A trigger that is defined with the trigger granularity FOR EACH ROW.

RRE. Residual recovery entry (in IMS).

RRSAF. Recoverable Resource Manager Services attachment facility.

RS. Read stability.

RTT. Resource translation table.

RURE. Restart URE.

S

savepoint. A named entity that represents the state of data and schemas at a particular point in time within a unit of work. SQL statements exist to set a savepoint, release a savepoint, and restore data and schemas to the state that the savepoint represents. The restoration of data and schemas to a savepoint is usually referred to as *rolling back to a savepoint*.

SBCS. Single-byte character set.

SCA. Shared communications area.

scalar function. An SQL operation that produces a
single value from another value and is expressed as a
function name, followed by a list of arguments that are
enclosed in parentheses. Contrast with *aggregate*
function.

scale. In SQL, the number of digits to the right of the decimal point (called the *precision* in the C language). The DB2 library uses the SQL definition.

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 ...
```

scrollability. The ability to use a cursor to fetch in either a forward or backward direction. The FETCH statement supports multiple fetch orientations to indicate the new position of the cursor. See also *fetch orientation*.

scrollable cursor. A cursor that can be moved in both a forward and a backward direction.

SDWA. System diagnostic work area.

search condition. A criterion for selecting rows from a table. A search condition consists of one or more predicates.

secondary authorization ID. An authorization ID that has been associated with a primary authorization ID by an authorization exit routine.

secondary group buffer pool. For a duplexed group buffer pool, the structure that is used to back up changed pages that are written to the primary group buffer pool. No page registration or cross-invalidation occurs using the secondary group buffer pool. The z/OS equivalent is *new* structure.

secondary index. A nonpartitioning index on a partitioned table.

section. The segment of a plan or package that contains the executable structures for a single SQL statement. For most SQL statements, one section in the plan exists for each SQL statement in the source program. However, for cursor-related statements, the DECLARE, OPEN, FETCH, and CLOSE statements reference the same section because they each refer to the SELECT statement that is named in the DECLARE CURSOR statement. SQL statements such as COMMIT, ROLLBACK, and some SET statements do not use a section.

segment. A group of pages that holds rows of a single table. See also *segmented table space*.

segmented table space. A table space that is divided into equal-sized groups of pages called segments. Segments are assigned to tables so that rows of different tables are never stored in the same segment.

self-referencing constraint. A referential constraint that defines a relationship in which a table is a dependent of itself.

self-referencing table. A table with a self-referencing constraint.

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.

sequential data set. A non-DB2 data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape. Several of the DB2 database utilities require sequential data sets.

sequential prefetch. A mechanism that triggers consecutive asynchronous I/O operations. Pages are fetched before they are required, and several pages are read with a single I/O operation.

serial cursor. A cursor that can be moved only in a forward direction.

serialized profile. A Java object that contains SQL statements and descriptions of host variables. The SQLJ translator produces a serialized profile for each connection context.

server. The target of a request from a remote requester. In the DB2 environment, the server function is provided by the distributed data facility, which is used to access DB2 data from remote applications.

server-side programming. A method for adding DB2 data into dynamic Web pages.

service class. An eight-character identifier that is used by the z/OS Workload Manager to associate user performance goals with a particular DDF thread or stored procedure. A service class is also used to classify work on parallelism assistants.

service request block. A unit of work that is scheduled to execute in another address space.

session. A link between two nodes in a VTAM network.

session protocols. The available set of SNA communication requests and responses.

shared communications area (SCA). A coupling facility list structure that a DB2 data sharing group uses for inter-DB2 communication.

share lock. A lock that prevents concurrently executing application processes from changing data, but not from reading data. Contrast with *exclusive lock*.

shift-in character. A special control character (X'0F') that is used in EBCDIC systems to denote that the subsequent bytes represent SBCS characters. See also *shift-out character*.

shift-out character. A special control character (X'0E') that is used in EBCDIC systems to denote that the subsequent bytes, up to the next shift-in control character, represent DBCS characters. See also *shift-in character*.

sign-on. A request that is made on behalf of an individual CICS or IMS application process by an attachment facility to enable DB2 to verify that it is authorized to use DB2 resources.

simple page set. A nonpartitioned page set. A simple page set initially consists of a single data set (page set piece). If and when that data set is extended to 2 GB, another data set is created, and so on, up to a total of 32 data sets. DB2 considers the data sets to be a single contiguous linear address space containing a maximum of 64 GB. Data is stored in the next available location within this address space without regard to any partitioning scheme.

simple table space. A table space that is neither partitioned nor segmented.

single-byte character set (SBCS). A set of characters in which each character is represented by a single byte. Contrast with *double-byte character set* or *multibyte character set*.

single-precision floating point number. A 32-bit approximate representation of a real number.

size. In the C language, the total number of digits in a decimal number (called the *precision* in SQL). The DB2 library uses the SQL term.

SMF. System Management Facilities.

SMP/E. System Modification Program/Extended.

SMS. Storage Management Subsystem.

SNA. Systems Network Architecture.

SNA network. The part of a network that conforms to the formats and protocols of Systems Network Architecture (SNA).

socket. A callable TCP/IP programming interface that TCP/IP network applications use to communicate with remote TCP/IP partners.

sourced function. A function that is implemented by another built-in or user-defined function that is already known to the database manager. This function can be a scalar function or a column (aggregating) function; it returns a single value from a set of values (for example, MAX or AVG). Contrast with *built-in function, external function,* and *SQL function*.

source program. A set of host language statements and SQL statements that is processed by an SQL precompiler.

source table. A table that can be a base table, a view, a table expression, or a user-defined table function.

source type. An existing type that DB2 uses to internally represent a distinct type.

space. A sequence of one or more blank characters.

special register. A storage area that DB2 defines for an application process to use for storing information that can be referenced in SQL statements. Examples of special registers are USER and CURRENT DATE.

specific function name. A particular user-defined function that is known to the database manager by its specific name. Many specific user-defined functions can have the same function name. When a user-defined function is defined to the database, every function is assigned a specific name that is unique within its schema. Either the user can provide this name, or a default name is used.

SPUFI. SQL Processor Using File Input.

SQL. Structured Query Language.

SQL authorization ID (SQL ID). The authorization ID that is used for checking dynamic SQL statements in some situations.

SQLCA. SQL communication area.

SQL communication area (SQLCA). A structure that is used to provide an application program with information about the execution of its SQL statements.

SQL connection. An association between an application process and a local or remote application server or database server.

SQLDA. SQL descriptor area.

SQL descriptor area (SQLDA). A structure that describes input variables, output variables, or the columns of a result table.

SQL escape character. The symbol that is used to enclose an SQL delimited identifier. This symbol is the double quotation mark ("). See also *escape character*.

SQL function. A user-defined function in which the CREATE FUNCTION statement contains the source code. The source code is a single SQL expression that evaluates to a single value. The SQL user-defined function can return only one parameter.

SQL ID. SQL authorization ID.

SQLJ. Structured Query Language (SQL) that is embedded in the Java programming language.

SQL path. An ordered list of schema names that are used in the resolution of unqualified references to user-defined functions, distinct types, and stored procedures. In dynamic SQL, the current path is found in the CURRENT PATH special register. In static SQL, it is defined in the PATH bind option.

SQL procedure. A user-written program that can be invoked with the SQL CALL statement. Contrast with *external procedure*.

SQL processing conversation. Any conversation that requires access of DB2 data, either through an application or by dynamic query requests.

SQL Processor Using File Input (SPUFI). A facility of the TSO attachment subcomponent that enables the DB2I user to execute SQL statements without embedding them in an application program.

SQL return code. Either SQLCODE or SQLSTATE.

SQL routine. A user-defined function or stored procedure that is based on code that is written in SQL.

SQL statement coprocessor. An alternative to the DB2 precompiler that lets the user process SQL statements at compile time. The user invokes an SQL statement coprocessor by specifying a compiler option.

SQL string delimiter. A symbol that is used to enclose an SQL string constant. The SQL string delimiter is the apostrophe ('), except in COBOL applications, where the user assigns the symbol, which is either an apostrophe or a double quotation mark (").

SRB. Service request block.

SSI. Subsystem interface (in z/OS).

SSM. Subsystem member (in IMS).

stand-alone. An attribute of a program that means that it is capable of executing separately from DB2, without using DB2 services.

star join. A method of joining a dimension column of a fact table to the key column of the corresponding dimension table. See also *join*, *dimension*, and *star schema*.

star schema. The combination of a fact table (which contains most of the data) and a number of dimension tables. See also *star join*, *dimension*, and *dimension table*.

statement handle. In DB2 ODBC, the data object that contains information about an SQL statement that is managed by DB2 ODBC. This includes information such as dynamic arguments, bindings for dynamic arguments and columns, cursor information, result values, and status information. Each statement handle is associated with the connection handle.

statement string. For a dynamic SQL statement, the character string form of the statement.

statement trigger. A trigger that is defined with the trigger granularity FOR EACH STATEMENT.

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.*

static SQL. SQL statements, embedded within a program, that are prepared during the program preparation process (before the program is executed). After being prepared, the SQL statement does not change (although values of host variables that are specified by the statement might change).

storage group. A named set of disks on which DB2 data can be stored.

stored procedure. A user-written application program that can be invoked through the use of the SQL CALL statement.

string. See *character string* or *graphic string*.

strong typing. A process that guarantees that only user-defined functions and operations that are defined on a distinct type can be applied to that type. For example, you cannot directly compare two currency types, such as Canadian dollars and U.S. dollars. But you can provide a user-defined function to convert one currency to the other and then do the comparison.

structure. (1) A name that refers collectively to different types of DB2 objects, such as tables, databases, views, indexes, and table spaces. (2) A construct that uses z/OS to map and manage storage on a coupling facility. See also *cache structure*, *list structure*, or *lock structure*.

Structured Query Language (SQL). A standardized language for defining and manipulating data in a relational database.

structure owner. In relation to group buffer pools, the DB2 member that is responsible for the following activities:

- Coordinating rebuild, checkpoint, and damage assessment processing
- Monitoring the group buffer pool threshold and notifying castout owners when the threshold has been reached

subcomponent. A group of closely related DB2 modules that work together to provide a general function.

subject table. The table for which a trigger is created. When the defined triggering event occurs on this table, the trigger is activated.

subpage. The unit into which a physical index page can be divided.

subquery. A SELECT statement within the WHERE or HAVING clause of another SQL statement; a nested SQL statement.

subselect. That form of a query that does not include an ORDER BY clause, an UPDATE clause, or UNION operators.

substitution character. A unique character that is substituted during character conversion for any characters in the source program that do not have a match in the target coding representation.

subsystem. A distinct instance of a relational database management system (RDBMS).

surrogate pair. A coded representation for a single character that consists of a sequence of two 16-bit code units, in which the first value of the pair is a high-surrogate code unit in the range U+D800 through U+DBFF, and the second value is a low-surrogate code unit in the range U+DC00 through U+DFFF. Surrogate pairs provide an extension mechanism for encoding 917 476 characters without requiring the use of 32-bit characters.

SVC dump. A dump that is issued when a z/OS or a DB2 functional recovery routine detects an error.

sync point. See *commit point*.

syncpoint tree. The tree of recovery managers and resource managers that are involved in a logical unit of work, starting with the recovery manager, that make the final commit decision.

synonym. In SQL, an alternative name for a table or view. Synonyms can be used to refer only to objects at the subsystem in which the synonym is defined.

syntactic character set. A set of 81 graphic characters that are registered in the IBM registry as character set 00640. This set was originally recommended to the programming language community to be used for syntactic purposes toward maximizing portability and interchangeability across systems and country boundaries. It is contained in most of the primary registered character sets, with a few exceptions. See also *invariant character set*.

Sysplex. See *Parallel Sysplex*.

Sysplex query parallelism. Parallel execution of a single query that is accomplished by using multiple tasks on more than one DB2 subsystem. See also *query CP parallelism*.

system administrator. The person at a computer installation who designs, controls, and manages the use of the computer system.

system agent. A work request that DB2 creates internally such as prefetch processing, deferred writes, and service tasks.

system conversation. The conversation that two DB2 subsystems must establish to process system messages before any distributed processing can begin.

system diagnostic work area (SDWA). The data that is recorded in a SYS1.LOGREC entry that describes a program or hardware error.

system-directed connection. A connection that a relational DBMS manages by processing SQL statements with three-part names.

System Modification Program/Extended (SMP/E). A z/OS tool for making software changes in programming systems (such as DB2) and for controlling those changes.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information through and controlling the configuration and operation of networks.

SYS1.DUMPxx data set. A data set that contains a system dump (in z/OS).

SYS1.LOGREC. A service aid that contains important information about program and hardware errors (in z/OS).

T

table. A named data object consisting of a specific number of columns and some number of unordered rows. See also *base table* or *temporary table*.

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.

table function. A function that receives a set of arguments and returns a table to the SQL statement that references the function. A table function can be referenced only in the FROM clause of a subselect.

table locator. A mechanism that allows access to trigger transition tables in the FROM clause of SELECT statements, in the subselect of INSERT statements, or from within user-defined functions. A table locator is a fullword integer value that represents a transition table.

table space. A page set that is used to store the records in one or more tables.

table space set. A set of table spaces and partitions that should be recovered together for one of these reasons:

- Each of them contains a table that is a parent or descendent of a table in one of the others.
- The set contains a base table and associated auxiliary tables.

A table space set can contain both types of relationships.

task control block (TCB). A z/OS control block that is used to communicate information about tasks within an address space that are connected to DB2. See also *address space connection*.

TB. Terabyte (1 099 511 627 776 bytes).

TCB. Task control block (in z/OS).

TCP/IP. A network communication protocol that computer systems use to exchange information across telecommunication links.

TCP/IP port. A 2-byte value that identifies an end user or a TCP/IP network application within a TCP/IP host.

template. A DB2 utilities output data set descriptor that is used for dynamic allocation. A template is defined by the TEMPLATE utility control statement.

temporary table. A table that holds temporary data. Temporary tables are useful for holding or sorting intermediate results from queries that contain a large number of rows. The two types of temporary table, which are created by different SQL statements, are the created temporary table and the declared temporary table. Contrast with *result table*. See also *created temporary table* and *declared temporary table*.

Terminal Monitor Program (TMP). A program that provides an interface between terminal users and command processors and has access to many system services (in z/OS).

thread. The DB2 structure that describes an application's connection, traces its progress, processes resource functions, and delimits its accessibility to DB2 resources and services. Most DB2 functions execute under a thread structure. See also *allied thread* and *database access thread*.

threadsafe. A characteristic of code that allows multithreading both by providing private storage areas for each thread, and by properly serializing shared (global) storage areas.

three-part name. The full name of a table, view, or alias. It consists of a location name, authorization ID, and an object name, separated by a period.

time. A three-part value that designates a time of day in hours, minutes, and seconds.

time duration. A decimal integer that represents a number of hours, minutes, and seconds.

timeout. Abnormal termination of either the DB2 subsystem or of an application because of the unavailability of resources. Installation specifications are set to determine both the amount of time DB2 is to wait for IRLM services after starting, and the amount of time IRLM is to wait if a resource that an application requests is unavailable. If either of these time specifications is exceeded, a timeout is declared.

Time-Sharing Option (TSO). An option in MVS that provides interactive time sharing from remote terminals.

timestamp. A seven-part value that consists of a date and time. The timestamp is expressed in years, months, days, hours, minutes, seconds, and microseconds.

TMP. Terminal Monitor Program.

to-do. A state of a unit of recovery that indicates that the unit of recovery's changes to recoverable DB2 resources are indoubt and must either be applied to the disk media or backed out, as determined by the commit coordinator.

trace. A DB2 facility that provides the ability to monitor and collect DB2 monitoring, auditing, performance, accounting, statistics, and serviceability (global) data.

transaction lock. A lock that is used to control concurrent execution of SQL statements.

transaction program name. In SNA LU 6.2 conversations, the name of the program at the remote logical unit that is to be the other half of the conversation.

transient XML data type. A data type for XML values that exists only during query processing.

transition table. A temporary table that contains all the affected rows of the subject table in their state before or after the triggering event occurs. Triggered SQL statements in the trigger definition can reference the table of changed rows in the old state or the new state.

transition variable. A variable that contains a column value of the affected row of the subject table in its state before or after the triggering event occurs. Triggered SQL statements in the trigger definition can reference the set of old values or the set of new values.

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.

trigger. A set of SQL statements that are stored in a DB2 database and executed when a certain event occurs in a DB2 table.

trigger activation. The process that occurs when the trigger event that is defined in a trigger definition is executed. Trigger activation consists of the evaluation of the triggered action condition and conditional execution of the triggered SQL statements.

trigger activation time. An indication in the trigger definition of whether the trigger should be activated before or after the triggered event.

trigger body. The set of SQL statements that is executed when a trigger is activated and its triggered action condition evaluates to true. A trigger body is also called *triggered SQL statements*.

trigger cascading. The process that occurs when the triggered action of a trigger causes the activation of another trigger.

triggered action. The SQL logic that is performed when a trigger is activated. The triggered action consists of an optional triggered action condition and a set of triggered SQL statements that are executed only if the condition evaluates to true.

triggered action condition. An optional part of the triggered action. This Boolean condition appears as a WHEN clause and specifies a condition that DB2 evaluates to determine if the triggered SQL statements should be executed.

triggered SQL statements. The set of SQL statements that is executed when a trigger is activated and its triggered action condition evaluates to true. Triggered SQL statements are also called the *trigger body*.

trigger granularity. A characteristic of a trigger, which determines whether the trigger is activated:
- Only once for the triggering SQL statement
- Once for each row that the SQL statement modifies

triggering event. The specified operation in a trigger definition that causes the activation of that trigger. The triggering event is comprised of a triggering operation (INSERT, UPDATE, or DELETE) and a subject table on which the operation is performed.

triggering SQL operation. The SQL operation that causes a trigger to be activated when performed on the subject table.

trigger package. A package that is created when a CREATE TRIGGER statement is executed. The package is executed when the trigger is activated.

TSO. Time-Sharing Option.

TSO attachment facility. A DB2 facility consisting of the DSN command processor and DB2I. Applications that are not written for the CICS or IMS environments can run under the TSO attachment facility.

typed parameter marker. A parameter marker that is specified along with its target data type. It has the general form:

```
CAST(? AS data-type)
```

type 1 indexes. Indexes that were created by a release of DB2 before DB2 Version 4 or that are specified as type 1 indexes in Version 4. Contrast with *type 2 indexes*. As of Version 8, type 1 indexes are no longer supported.

type 2 indexes. Indexes that are created on a release of DB2 after Version 7 or that are specified as type 2 indexes in Version 4 or later.

U

UCS-2. Universal Character Set, coded in 2 octets, which means that characters are represented in 16-bits per character.

UDF. User-defined function.

UDT. User-defined data type. In DB2 UDB for z/OS, the term *distinct type* is used instead of user-defined data type. See *distinct type*.

uncommitted read (UR). The isolation level that allows an application to read uncommitted data.

underlying view. The view on which another view is directly or indirectly defined.

undo. A state of a unit of recovery that indicates that the changes that the unit of recovery made to recoverable DB2 resources must be backed out.

Unicode. A standard that parallels the ISO-10646 standard. Several implementations of the Unicode standard exist, all of which have the ability to represent a large percentage of the characters that are contained in the many scripts that are used throughout the world.

uniform resource locator (URL). A Web address, which offers a way of naming and locating specific items on the Web.

union. An SQL operation that combines the results of two SELECT statements. Unions are often used to merge lists of values that are obtained from several tables.

unique constraint. An SQL rule that no two values in a primary key, or in the key of a unique index, can be the same.

unique index. An index that ensures that no identical key values are stored in a column or a set of columns in a table.

unit of recovery. A recoverable sequence of operations within a single resource manager, such as an instance of DB2. Contrast with *unit of work*.

unit of recovery identifier (URID). The LOGRBA of the first log record for a unit of recovery. The URID also appears in all subsequent log records for that unit of recovery.

unit of work. A recoverable sequence of operations within an application process. At any time, an application process is a single unit of work, but the life of an application process can involve many units of work as a result of commit or rollback operations. In a *multisite update* operation, a single unit of work can include several *units of recovery*. Contrast with *unit of recovery*.

Universal Unique Identifier (UUID). An identifier that is immutable and unique across time and space (in z/OS).

unlock. The act of releasing an object or system resource that was previously locked and returning it to general availability within DB2.

untyped parameter marker. A parameter marker that is specified without its target data type. It has the form of a single question mark (?).

updatability. The ability of a cursor to perform positioned updates and deletes. The updatability of a cursor can be influenced by the SELECT statement and the cursor sensitivity option that is specified on the DECLARE CURSOR statement.

update hole. The location on which a cursor is positioned when a row in a result table is fetched again and the new values no longer satisfy the search condition. DB2 marks a row in the result table as an update hole when an update to the corresponding row in the database causes that row to no longer qualify for the result table.

update trigger. A trigger that is defined with the triggering SQL operation UPDATE.

upstream. The node in the syncpoint tree that is responsible, in addition to other recovery or resource managers, for coordinating the execution of a two-phase commit.

UR. Uncommitted read.

URE. Unit of recovery element.

URID . Unit of recovery identifier.

URL. Uniform resource locator.

user-defined data type (UDT). See *distinct type*.

user-defined function (UDF). A function that is defined to DB2 by using the CREATE FUNCTION statement and that can be referenced thereafter in SQL statements. A user-defined function can be an *external function,* a *sourced function,* or an *SQL function.* Contrast with *built-in function*.

user view. In logical data modeling, a model or representation of critical information that the business requires.

UTF-8. Unicode Transformation Format, 8-bit encoding form, which is designed for ease of use with existing ASCII-based systems. The CCSID value for data in UTF-8 format is 1208. DB2 UDB for z/OS supports UTF-8 in mixed data fields.

UTF-16. Unicode Transformation Format, 16-bit encoding form, which is designed to provide code values for over a million characters and a superset of UCS-2. The CCSID value for data in UTF-16 format is 1200. DB2 UDB for z/OS supports UTF-16 in graphic data fields.

UUID. Universal Unique Identifier.

V

value. The smallest unit of data that is manipulated in SQL.

variable. A data element that specifies a value that can be changed. A COBOL elementary data item is an example of a variable. Contrast with *constant*.

variant function. See *nondeterministic function*.

varying-length string. A character or graphic string whose length varies within set limits. Contrast with *fixed-length string*.

version. A member of a set of similar programs, DBRMs, packages, or LOBs.

A version of a program is the source code that is produced by precompiling the program. The program version is identified by the program name and a timestamp (consistency token).

A version of a DBRM is the DBRM that is produced by precompiling a program. The DBRM version is identified by the same program name and timestamp as a corresponding program version.

A version of a package is the result of binding a DBRM within a particular database system. The package version is identified by the same program name and consistency token as the DBRM.

A version of a LOB is a copy of a LOB value at a point in time. The version number for a LOB is stored in the auxiliary index entry for the LOB.

view. An alternative representation of data from one or more tables. A view can include all or some of the columns that are contained in tables on which it is defined.

view check option. An option that specifies whether every row that is inserted or updated through a view must conform to the definition of that view. A view check option can be specified with the WITH CASCADED CHECK OPTION, WITH CHECK OPTION, or WITH LOCAL CHECK OPTION clauses of the CREATE VIEW statement.

Virtual Storage Access Method (VSAM). An access method for direct or sequential processing of fixed- and varying-length records on disk devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by relative-record number (in z/OS).

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in an SNA network (in z/OS).

| **volatile table.** A table for which SQL operations
| choose index access whenever possible.

VSAM. Virtual Storage Access Method.

VTAM. Virtual Telecommunication Access Method (in z/OS).

W

warm start. The normal DB2 restart process, which involves reading and processing log records so that data that is under the control of DB2 is consistent. Contrast with *cold start*.

WLM application environment. A z/OS Workload Manager attribute that is associated with one or more stored procedures. The WLM application environment determines the address space in which a given DB2 stored procedure runs.

write to operator (WTO). An optional user-coded service that allows a message to be written to the system console operator informing the operator of errors and unusual system conditions that might need to be corrected (in z/OS).

WTO. Write to operator.

WTOR. Write to operator (WTO) with reply.

X

XCF. See *cross-system coupling facility*.

XES. See *cross-system extended services*.

| **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.

X/Open. An independent, worldwide open systems organization that is supported by most of the world's largest information systems suppliers, user organizations, and software companies. X/Open's goal is to increase the portability of applications by combining existing and emerging standards.

XRF. Extended recovery facility.

Z

| **z/OS.** An operating system for the eServer™ product
| line that supports 64-bit real and virtual storage.

z/OS Distributed Computing Environment (z/OS DCE). A set of technologies that are provided by the Open Software Foundation to implement distributed computing.

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 Communications Server: IP and SNA Codes*, SC31-8791
- *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

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Program Number: 5625-DB2

Printed in USA

SC18-7427-07