DB2 Universal Database for OS/390

Utility Guide and Reference

Version 6

SC26-9015-01

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix E, “Notices” on page 545.

Second Edition, Softcopy Only (April 2000)

This edition applies to Version 6 of DB2 Universal Database Server for OS/390, 5645-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 manual since the hardcopy manual was published are indicated by the hash (#) symbol in the left-hand margin. Editorial changes that have no technical significance are not noted.

Contents

Section 1. Introduction

Chapter 1-1. Introduction to this book and the DB2® for OS/390® library

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

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. Familiarity with DB2 for OS/390 is recommended prior to using this book.

How to use this book

It is assumed that you possess an understanding of system administration, database administration, system operation, or application programming in the DB2 environment, as provided by the appropriate guide, and that you have some knowledge of the following:

- One of the transaction managers (CICS®, IMS™), or TSO
- A programming language (Assembler language, PL/I, COBOL, APL2®, BASIC, Fortran, Prolog, or C)
- OS/VS MVS Job Control Language (JCL)
- Structured Query Language (SQL)

This book contains the following sections and appendixes:

Product terminology and citations

In this book, DB2 Universal Database Server for OS/390 is referred to as "DB2 for OS/390." In cases where the context makes the meaning clear, DB2 for OS/390 is referred to as "DB2." When this book refers to other books in this library, a short title is used. (For example, "See *DB2 SQL Reference*" is a citation to *IBM DATABASE 2 Universal Database Server for OS/390 SQL Reference*.)

References in this book to "DB2 UDB" relate to the DB2 Universal Database™ product that is available on the AIX®, OS/2®, and Windows NT™ operating

systems. When this book refers to books about the DB2 UDB product, the citation includes the complete title and order number.

The following terms are used as indicated:

DB2® Represents either the DB2 licensed program or a particular DB2 subsystem.

C and C language Represent the C programming language.

CICS® Represents CICS/ESA® and CICS Transaction Server for OS/390 Release 1.

IMS™ Represents IMS/ESA®.

MVS Represents the MVS element of OS/390.

How to read the syntax diagrams

The following rules apply to the syntax diagrams 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.

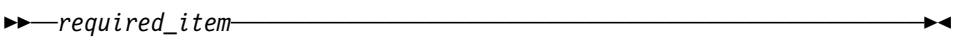
  The —► symbol indicates that the statement syntax is continued on the next line.

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

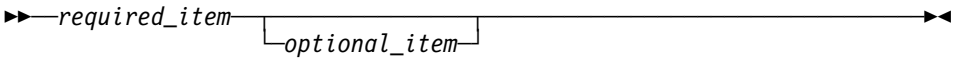
  The —►◄ symbol indicates the end of a statement.

  Diagrams of syntactical units other than complete statements start with the ►— symbol and end with the —► symbol.

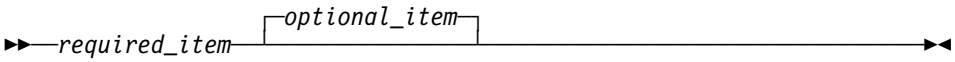
- Required items appear on the horizontal line (the main path).



- Optional items appear below the main path.

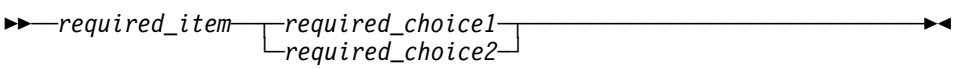


  If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

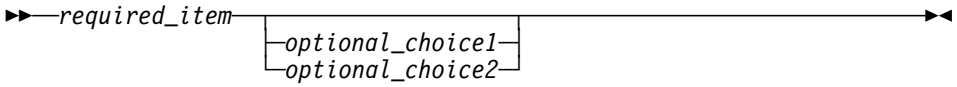


- If you can choose from two or more items, they appear vertically, in a stack.

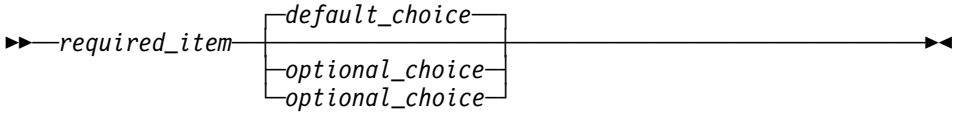
  If you *must* choose one of the items, one item of the stack appears on the main path.



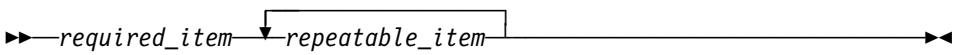
  If choosing one of the items is optional, the entire stack appears below the main path.



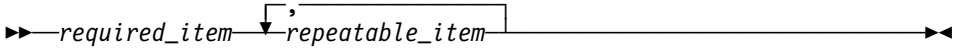
If one of the items is the default, it appears above the main path and the remaining choices are shown below.



- 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.

What naming conventions are used

This section describes naming conventions unique to commands and utilities.

When a parameter refers to an object created by SQL statements (for example, tables, table spaces, and indexes), follow the SQL syntactical naming conventions.

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 reserved as alphabetic extenders for national languages, #, @, and $ in the United States).
- 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 3 of *DB2 SQL Reference* for an additional explanation of long identifiers, short identifiers, and location identifiers.

authorization-id A short identifier of 1 to 8 letters, digits, or the underscore that identifies a set of privileges. An authorization ID must begin with a letter.

connection-name

An identifier of 1 to 8 characters that identifies an address space connection to DB2. A connection identifier is one of the following:

- For DSN processes running in TSO foreground, the connection name TSO is used.
- For DSN processes running in TSO batch, the connection name BATCH is used.
- For the call attachment facility (CAF), the connection name DB2CALL is used.
- For IMS and CICS processes, the connection name is the system identification name.

See Section 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:

- For DSN processes running in TSO foreground, the correlation ID is the TSO logon identifier.
- For DSN processes running in TSO batch, the correlation ID is the job name.
- For CAF processes, the correlation ID is the TSO logon identifier.
- For IMS processes, the correlation ID is the PST#.PSBNAME.
- For CICS processes, the correlation ID is the entry identifier.thread_number.transaction_identifier.

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

database-name

A short identifier that designates 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 1 to 8 letters or digits that identifies a member of a partitioned data set.

A DBRM member name should not begin with DSN; this can sometimes conflict with DB2-provided DBRM member names. If a DBRM member name beginning with DSN is specified, 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 1 to 8 characters that designates 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 an apostrophe. The characters between the string delimiters must be a hexadecimal number.

index-name A qualified or unqualified name that designates 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 set forth in Chapter 3 of *DB2 SQL Reference*.

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 data base management system. A location name must begin with a letter.

luname An SQL short identifier of 1 to 8 characters that identifies a logical unit name. An luname must begin with a letter.

member-name An identifier of 1 to 8 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; this can sometimes conflict with DB2-provided member names. If a member name beginning with DSN is specified, DB2 issues a warning message.

qualifier-name An SQL short identifier of 1 to 8 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 MVS.

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

A fully qualified table name is a three-part name. The first part is a location name that designates the DBMS at which the table is stored. The second part is the authorization ID that designates 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 designates the owner of the table. The second part is a 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 is an authorization ID, which is determined by the rules set forth in Chapter 3 of *DB2 SQL Reference*.

table-space-name A short identifier that designates 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 upper and lower case letters, numbers 0 through 9, and the following characters: #, $, @, ¢, !, ¬, and ..

How to use the DB2 library

Titles of books in the library begin with DB2 Universal Database for OS/390 Version 6. 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.

Throughout the library, the DB2 for OS/390 licensed program and a particular DB2 for MVS/ESA subsystem are each referred to as "DB2." In each case, the context makes the meaning clear.

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 associated with DB2 are grouped into the following major categories (but supplemental information relating to all of the below tasks for new releases of DB2 can be found in *DB2 Release 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 then you also need *DB2 Data Sharing: Planning and Administration*, which describes installation considerations for data sharing.

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 Query Management Facility (QMF™) or some other program, and the library for that program might provide all the instruction or reference material they need. For a list of the titles in the 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 *DB2 Application*

Programming and SQL Guide, *DB2 SQL Reference*, and *DB2 ODBC Guide and Reference* just as end users do.

Application programmers also need instructions on many other topics:

- How to transfer data between DB2 and a host program—written in COBOL, C, or FORTRAN, 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 platforms
- How to write applications that use DB2 ODBC to access DB2 servers
- 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 Messages and Codes* .

Information about writing applications across platforms can be found in *Distributed Relational Database Architecture™: Application Programming Guide*.

System and Database Administration: *Administration* covers almost everything else. *DB2 Administration Guide* divides those tasks among the following sections:

- Section 2 (Volume 1) of *DB2 Administration Guide* discusses the decisions that must be made when designing a database and tells how to bring the design into being by creating DB2 objects, loading data, and adjusting to changes.
- Section 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.
- Section 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.
- Section 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.

In addition, the appendixes in *DB2 Administration Guide* contain valuable information on DB2 sample tables, National Language Support (NLS), writing exit routines, interpreting DB2 trace output, and character conversion for distributed data.

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, then 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, then 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 Codes*, which lists 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* and *DB2 Messages and Codes*.

How to obtain DB2 information

DB2 on the Web

Stay current with the latest information about DB2. View the DB2 home page on the World Wide Web. News items keep you informed about the latest enhancements to the product. Product announcements, press releases, fact sheets, and technical articles help you plan your database management strategy.

You can view and search DB2 publications on the Web, or you can download and print many of the most current DB2 books. Follow links to other Web sites with more information about DB2 family and OS/390 solutions. Access DB2 on the Web at the following address:

http://www.ibm.com/software/db2os390

DB2 publications

The DB2 publications for DB2 Universal Database Server for OS/390 are available in both hardcopy and softcopy format.

BookManager® format

Using online books on CD-ROM, you can read, search across books, print portions of the text, and make notes in these BookManager books. With the appropriate BookManager READ product or IBM Library Readers, you can view these books in the OS/390, VM, OS/2, DOS, AIX, and Windows™ environments. You can also view many of the DB2 BookManager books on the Web.

PDF format

Many of the DB2 books are available in Portable Document Format (PDF) for viewing or printing from CD-ROM or the Web. Download the PDF books to your intranet for distribution throughout your enterprise.

CD-ROMs

Books for Version 6 of DB2 Universal Database Server for OS/390 are available on CD-ROMs:

- *DB2 UDB for OS/390 Version 6 Licensed Online Book*, LK3T-3519, containing *DB2 UDB for OS/390 Version 6 Diagnosis Guide and Reference* in BookManager format, for ordering with the product.
- *DB2 UDB Server for OS/390 Version 6 Online and PDF Library*, SK3T-3518, a collection of books for the DB2 server in BookManager and PDF formats.

Periodically, the books will be refreshed on subsequent editions of these CD-ROMs.

The books for Version 6 of DB2 UDB Server for OS/390 are also available on the following collection kits that contain online books for many IBM products:

- *Online Library Omnibus Edition OS/390 Collection*, SK2T-6700, in English
- *IBM Online Library MVS Collection Kit*, SK88-8002, in Japanese, for viewing on DOS and Windows operating systems.

DB2 education

IBM Education and Training offers a wide variety of classroom courses to help you quickly and efficiently gain DB2 expertise. Classes are scheduled in cities all over the world. You can find class information, by country, at the IBM Learning Services Web site:

http://www.ibm.com/services/learning/

For more information, including the current local schedule, please contact your IBM representative.

Classes can also be taught at your location, at a time that suits your needs. Courses can even be customized to meet your exact requirements. The *All-in-One Education and Training Catalog* describes the DB2 curriculum in the United States. You can inquire about or enroll in these courses by calling 1-800-IBM-TEACH (1-800-426-8322).

How to order the DB2 library

You can order DB2 publications and CD-ROMs through your IBM representative or the IBM branch office serving your locality. If you are located within the United States or Canada, you can place your order by calling one of the toll-free numbers :

- In the U.S., call 1-800-879-2755.
- In Canada, call 1-800-565-1234.

To order additional copies of licensed publications, specify the SOFTWARE option. To order additional publications or CD-ROMs, specify the PUBLICATIONS and SLSS option. Be prepared to give your customer number, the product number, and the feature code(s) or order numbers you want.

Summary of changes to DB2 UDB for OS/390 Version 6

DB2 UDB for OS/390 Version 6 delivers an enhanced relational database server solution for OS/390. This release focuses on greater capacity, performance improvements for utilities and queries, easier database management, more powerful network computing, and DB2 family compatibility with rich new object-oriented capability, triggers, and more built-in functions.

Capacity improvements

16-terabyte tables provide a significant increase to table capacity for partitioned and LOB table spaces and indexes, and for nonpartitioning indexes.

Buffer pools in data spaces provide virtual storage constraint relief for the ssnmDBM1 address space, and data spaces increase the maximum amount of virtual buffer pool space allowed.

Performance and availability

Improved partition rebalancing lets you redistribute partitioned data with minimal impact to data availability. One REORG of a range of partitions both reorganizes and rebalances the partitions.

You can **change checkpoint frequency dynamically** using the new SET LOG command and initiate checkpoints any time while your subsystem remains available.

Utilities that are faster, more parallel, easier to use:

- **Faster backup and recovery** enables COPY and RECOVER to process a list of objects in parallel, and recover indexes and table spaces at the same time from image copies and the log.
- **Parallel index build** reduces the elapsed time of LOAD and REORG jobs of table spaces, or partitions of table spaces, that have more than one index; the elapsed time of REBUILD INDEX jobs is also reduced.
- Tests show **decreased elapsed and processor time for online REORG**.
- **Inline statistics** embeds statistics collection into utility jobs, making table spaces available sooner.
- You can **determine when to run REORG** by specifying threshold limits for relevant statistics from the DB2 catalog.

Query performance enhancements include:

- **Query parallelism extensions** for complex queries, such as outer joins and queries that use nonpartitioned tables
- **Improved workload balancing in a Parallel Sysplex®** that reduces elapsed time for a single query that is split across active DB2 members
- **Improved data transfer** that lets you request multiple DRDA query blocks when performing high-volume operations
- The ability to use an **index to access predicates with noncorrelated IN subqueries**
- **Faster query processing** of queries that include join operations

More performance and availability enhancements include:

- **Faster restart and recovery** with the ability to postpone backout work during restart, and a faster log apply process
- **Increased flexibility with 8-KB and 16-KB page sizes** for balancing different workload requirements more efficiently, and for controlling traffic to the coupling facility for some workloads
- **Direct-row access** using the new ROWID data type to re-access a row directly without using the index or scanning the table
- **Ability to retain prior access path** when you rebind a statement. You almost always get the same or a better access path. For the exceptional cases, Version 6 of DB2 for OS/390 lets you retain the access path from a prior BIND by using rows in an Explain table as input to optimization.
- An **increased log output buffer size** (from 1000 4-KB to 100000 4-KB buffers) that improves log read and write performance

Data sharing enhancements

More caching options use the coupling facility to improve performance in a data sharing environment for some applications by writing changed pages directly to DASD.

Control of space map copy maintenance with a new option avoids tracking of page changes, thereby optimizing performance of data sharing applications.

User productivity

Predictive governing capabilities enhance the resource limit facility to help evaluate resource consumption for queries that run against large volumes of data.

Statement cost estimation of processing resource that is needed for an SQL statement helps you to determine error and warning thresholds for governing, and to decide which statements need tuning.

A **default buffer pool** for user data and indexes isolates user data from the DB2 catalog and directory, and separating user data from system data helps you make better tuning decisions.

More information available for monitoring DB2 includes data set I/O activity in traces, both for batch reporting and online monitors.

Better integration of DB2 and Workload Manager delay reporting enables DB2 to notify Workload Manager about the current state of a work request.

More tables are allowed in SQL statements SELECT, UPDATE, INSERT, and DELETE, and in views. DB2 increases the limit from 15 to 225 tables. The number of tables and views in a subselect is not changed.

Improved DB2 UDB family compatibility includes SQL extensions, such as:

- A VALUES clause of INSERT that supports any expression
- A new VALUES INTO statement

Easier recovery management lets you achieve a single point of recovery and recover data at a remote site more easily.

Enhanced database commands extend support for pattern-matching characters (*) and let you filter display output.

You can easily **process dynamic SQL in batch mode** with the new object form of DSNTEP2 shipped with DB2 for OS/390.

Network computing

SQLJ, the newest Java implementation for the OS/390 environment, supports SQL embedded in the Java programming language. With SQLJ, your Java programs benefit from the superior performance, manageability, and authorization available to static SQL, and they are easy to write.

DRDA® support for three-part names offers more functionality to applications using three-part names for remote access and improves the performance of client/server applications.

Stored procedure enhancements include the ability to create and modify stored procedure definitions, make nested calls for stored procedures and user-defined functions, and imbed CALL statements in application programs or dynamically invoke CALL statements from IBM's ODBC and CLI drivers.

DB2 ODBC extensions include new and modified APIs and new data types to support the object-relational extensions.

ODBC access to DB2 for OS/390 catalog data improves the performance of your ODBC catalog queries by redirecting them to shadow copies of DB2 catalog tables.

Better performance for ODBC applications reduces the number of network messages that are exchanged when an application executes dynamic SQL.

Improvements for dynamically prepared SQL statements include a new special register that you use to implicitly qualify names of distinct types, user-defined functions, and stored procedures.

DDF connection pooling uses a new type of inactive thread that improves performance for large volumes of inbound DDF connections.

Object-relational extensions and active data

The object extensions of DB2 offer the benefits of object-oriented technology while increasing the strength of your relational database with an enriched set of data types and functions. Complementing these extensions is a powerful mechanism, triggers, that brings application logic into the database that governs the following new structures:

- **Large objects (LOBs)** are well suited to represent large, complex structures in DB2 tables. Now you can make effective use of multimedia by storing objects such as complex documents, videos, images, and voice. Some key elements of LOB support include:
  - LOB data types for storing byte strings up to 2 GB in size
  - LOB locators for easily manipulating LOB values in manageable pieces
  - Auxiliary tables (that reside in LOB table spaces) for storing LOB values

- **Distinct types** (which are sometimes called user-defined data types), like built-in data types, describe the data that is stored in columns of tables where the instances (or objects) of these data types are stored. They ensure that only those functions and operators that are explicitly defined on a distinct type can be applied to its instances.
- **User-defined functions**, like built-in functions or operators, support manipulation of distinct type instances (and built-in data types) in SQL queries.
- **New and extended built-in functions** improve the power of the SQL language with about 100 new built-in functions, extensions to existing functions, and sample user-defined functions.

Triggers automatically execute a set of SQL statements whenever a specified event occurs. These statements validate and edit database changes, read and modify the database, and invoke functions that perform operations inside and outside the database.

You can use the **DB2 Extenders** feature of DB2 for OS/390 to store and manipulate image, audio, video, and text objects. The extenders automatically capture and maintain object information and provide a rich body of APIs.

More function

Some function and capability is available to both Version 6 and Version 5 users. Learn how to obtain these functions now, prior to migrating to Version 6, by visiting the following Web site:

http://www.software.ibm.com/data/db2/os390/v5apar.html

Features of DB2 for OS/390

DB2 for OS/390 Version 6 offers a number of tools, which are optional features of the server, that are shipped to you automatically when you order DB2 Universal Database for OS/390:

- DB2 Management Tools Package, which includes the following elements:
  - DB2 UDB Control Center
  - DB2 Stored Procedures Builder
  - DB2 Installer
  - DB2 Visual Explain
  - DB2 Estimator
- Net.Data® for OS/390

You can install and use these features in a “Try and Buy” program for up to 90 days without paying license charges:

- Query Management Facility
- DB2 DataPropagator™
- DB2 Performance Monitor
- DB2 Buffer Pool Tool
- DB2 Administration Tool

Migration considerations

Migration to Version 6 eliminates all type 1 indexes, shared read-only data, data set passwords, use of host variables without the colon, and RECOVER INDEX usage. You can migrate to Version 6 only from a Version 5 subsystem.

Summary of changes to this book

The syntax diagram for the DSNU CLIST command has changed. See "DSNU CLIST command syntax" on page 35 for more information.

The following online utilities are new for Version 6:

"Chapter 2-6. CHECK LOB" on page 79
"Chapter 2-13. REBUILD INDEX" on page 209

The following online utilities have changed:

"Chapter 2-3. CATMAINT" on page 51
"Chapter 2-4. CHECK DATA" on page 55
"Chapter 2-5. CHECK INDEX" on page 71
"Chapter 2-7. COPY" on page 85
"Chapter 2-9. LOAD" on page 121
"Chapter 2-12. QUIESCE" on page 201
"Chapter 2-14. RECOVER" on page 225
"Chapter 2-15. REORG INDEX" on page 255
"Chapter 2-16. REORG TABLESPACE" on page 277
"Chapter 2-17. REPAIR" on page 343
"Chapter 2-18. REPORT" on page 365

The following stand-alone utilities have changed:

"Chapter 3-3. DSNJU003 (Change Log Inventory)" on page 411
"Chapter 3-6. DSN1COMP" on page 447
"Chapter 3-7. DSN1COPY" on page 455
"Chapter 3-9. DSN1PRNT" on page 493

The following appendixes are new for Version 6:

- Appendix B, "Invoking utilities as a stored procedure (DSNUTILS)" on page 517
- Appendix C, "Resetting an advisory or restrictive status" on page 527
- Appendix D, "How to run sample programs DSNTIAUL, DSNTIAD, and DSNTEP2" on page 535

Chapter 1-2. Introduction to the DB2 utilities

This chapter provides an introduction to the DB2 online and stand-alone utilities. This chapter also discusses the authorization rules for coding utility control statements and the data sets used by utilities.

Types of DB2 utilities

There are two types of DB2 utilities: online utilities and stand-alone utilities. Both types are described in this section.

Description of online utilities

DB2 online utilities run as standard MVS batch jobs or stored procedures, and they require DB2 to be running. They do not run under control of the terminal monitor program (TMP), but have their own attach mechanisms. They invoke DB2 control facility services directly. Refer to “Chapter 2-1. Invoking DB2 online utilities” on page 27 for information about the ways to execute these utilities.

Description of stand-alone utilities

The stand-alone utilities execute as batch jobs independent of DB2. They can be executed only by means of MVS JCL. Refer to the chapters on the individual utilities in “Chapter 3-1. Invoking stand-alone utilities” on page 407 for information about the ways to execute these utilities.

Privileges and authorization IDs

The issuer of a command or a utility job can be an individual user. It can also be a program running in batch mode or an IMS or CICS transaction. We use the term *process* to represent any or all of those.

A process is represented to DB2 by a set of identifiers (IDs). What the process can do with DB2 is determined by *privileges* and *authorities* that can be held by its identifiers. We use *“the privilege set of a process”* to mean 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 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 available to the process. A secondary authorization ID is often a 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 entered from an MVS console are not associated with any secondary authorization IDs.

- An SQL authorization ID (SQL ID) holds the privileges exercised when issuing certain dynamic SQL statements. This ID plays little part in the utilities described in this book.

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 "Processing connections" and "Processing sign-ons" in Section 3 (Volume 1) of *DB2 Administration Guide*.

A privilege or authority is granted to, or revoked from, an identifier by executing an SQL GRANT or a REVOKE statement. For the complete syntax of those statements, see Chapter 6 of *DB2 SQL Reference*.

If you use the access control authorization exit, then that exit might control the authorization rules, rather than the exits documented for each utility.

Objects supported by DB2 utilities

DB2 utilities operate on a variety of target 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. 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.

Running utilities when using undefined data sets

With DB2 version 6, you can run certain online utilities on table spaces or index spaces that were defined with the DEFINE NO attribute. In this situation, the physical creation of the data set is deferred until data is first inserted in the table space.

You can populate table spaces with undefined data sets by using the LOAD utility with either the RESUME keyword, the REPLACE keyword, or both. Using LOAD in this manner 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.

The following online utilities will process a table space that has been defined but has indexes that remain undefined:

- CHECK DATA
- CHECK INDEX
- COPY TABLESPACE
- MERGECOPY
- MODIFY
- QUIESCE WRITE(NO)
- REBUILD INDEX
- RECOVER

- REORG TABLESPACE
- REPAIR, except REPAIR DBD
- RUNSTATS TABLESPACE INDEX(ALL)

All online utilities except LOAD, REPAIR DBD, REPORT, and STOSPACE terminate processing after encountering an undefined target object by issuing message DSNU185I with return code 8.

No stand-alone utilities can be used on objects where the data sets have not been defined.

Section 2. DB2 online utilities

Chapter 2-1. 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 required to run an online utility.

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

- "Using the DB2 utilities panel in DB2I" on page 32
- "Using the DSNU CLIST command in TSO" on page 34
- "Using the supplied JCL procedure (DSNUPROC)" on page 41
- "Creating the JCL data set yourself" on page 43
- Appendix B, "Invoking utilities as a stored procedure (DSNUTILS)" on page 517

For the least involvement with JCL, use either DB2I or the DSNU CLIST command, 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 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 listed previously.

Creating utility control statements

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

You can create the utility control statements with the ISPF/PDF edit function. After they are created, save them in a sequential or partitioned data set.

Control statement coding rules

Utility control statements are read from the SYSIN input stream. The statements in that stream must obey these rules:

- If the SYSIN records are 80-character fixed-length records, columns 73 through 80 are ignored.
- The records are concatenated before being parsed; therefore, a statement or any of its syntactical constructs can span more than one record. No continuation character is necessary.
- The SYSIN stream must begin with one of these *online utility names*:

CATMAINT	LOAD	REORG
CHECK DATA	MERGECOPY	REPAIR
CHECK INDEX	MODIFY	REPORT
CHECK LOB	QUIESCE	RUNSTATS
COPY	REBUILD INDEX	STOSPACE
DIAGNOSE	RECOVER	

At least one blank character must follow the name.

- 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 online utility name determines which options can follow it. You can specify more than one utility control statement in the SYSIN stream.

Options are typically described by an *option keyword*, followed by a *parameter.* The parameter value can be a keyword. Values of parameters are sometimes enclosed in parentheses. The syntax diagrams for utility control statements included in this chapter show parentheses where they are required.

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

- The two hyphens must be on the same line, not separated by a space
- You can start comments wherever a space is valid, except within a delimiter token.
- Comments are terminated by the end of the line, for example:

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

Example of option description

Where the syntax of each utility control statement is described, parameters are indented under the option keyword they must follow. Here 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 parameter. As noted previously, you can enclose parameter values in parentheses but they are not always required. The description of the temporary work file can be written as either `WORKDDN SYSUT1` or `WORKDDN (SYSUT1)`.

Data sets used by online utilities

Every online utility job requires a SYSIN DD statement to describe an input data set; some utilities also require other data sets. Table 1 on page 29 lists the name of each DD statement that might be needed, the online utilities that require it, and the purpose of the corresponding data sets. If an alternate DD statement is allowed, you specify it as a parameter in a utility option. Table 1 on page 29 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 1 (Page 1 of 2). Data sets used by online utilities

DD Name	Used By	Purpose	Option Keyword
ddname	COPY[8]	A single data set DB2 uses when you specify the FILTERDDN option in the utility control statement; contains a list of VSAM data set names used during COPY using the CONCURRENT and FILTERDDN options.	FILTERDDN
DATAWK*nn*	REORG	Work data set for sorting data, where *nn* is a 2-digit number. You can use several data sets. To estimate the size of the data set needed, see "Data sets used by REORG TABLESPACE" on page 307.	NOSYSREC CHANGE
DSSPRINT	COPY	Output data set for messages; required when CONCURRENT copy is used and the SYSPRINT DD card points to a data set.	CONCURRENT
SORTOUT	CHECK DATA[2,7], LOAD[1,3,7], REORG[1,7]	Holds sorted keys (sort output) and allows the SORT phase to be restarted; for CHECK DATA, holds sorted keys (sort output).	WORKDDN
SORTWK*nn* [4]	CHECK DATA, CHECK INDEX, LOAD, REBUILD INDEX, REORG	Work data set for sorting indexes where *nn* is a 2-digit number. You can use several data sets. To estimate the size of the data set required, see page 64 for CHECK DATA, 74 for CHECK INDEX, 150 for LOAD, 214 for REBUILD INDEX, or 309 for REORG.	None
SW*mm*WK*nn* [4]	LOAD, REBUILD INDEX, REORG	Optional work data sets for sorting index keys using the SORTKEYS keyword, where *mm* and *nn* are 2-digit numbers. You can use several data sets. To estimate the size of the data set required, see page 167 for LOAD, 218 for REBUILD INDEX, or 323 for REORG.	None
SYSCOPY	COPY, MERGECOPY, LOAD[5], REORG [5]	An output data set for copies.	COPYDDN RECOVERYDDN
SYSDISC	LOAD, REORG DISCARD, optional for REORG	Contains discarded records (optional).	DISCARDDN
SYSERR	CHECK DATA[2], LOAD	Contains information about errors encountered during processing.	ERRDDN
SYSIN	All utilities	An input data set for utility statements.	None
SYSMAP	LOAD[3]	Contains information about what input records violated a constraint.	MAPDDN
SYSPRINT	All utilities	Messages and printed output (usually SYSOUT).	None

Table 1 (Page 2 of 2). Data sets used by online utilities

DD Name	Used By	Purpose	Option Keyword
SYSPUNCH	REORG	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.	PUNCHDDN
SYSREC	LOAD[2], REORG[6]	Contains the LOAD input data set; unloaded records for REORG.	INDDN UNLDDN
SYSUT1	CHECK DATA[7], CHECK INDEX[2], LOAD[1,3,7], MERGECOPY, REBUILD INDEX, REORG[1,7]	A temporary work data set that holds sorted keys for input to the SORT phase; for MERGECOPY, it holds intermediate merged output.	WORKDDN
UTPRIN*mm*	LOAD, REBUILD INDEX, REORG	Optional print message data sets, used when the SORTKEYS keyword is specified, where *mm* is a 2-digit number.	None
UTPRINT	CHECK DATA, CHECK INDEX, LOAD, REORG, REBUILD INDEX	Contains messages from DFSORT™ (usually, SYSOUT or DUMMY).	None

Notes:

[1] Required for tables with indexes, not required when using REORG with the SORTKEYS option.
[2] Required.
[3] When referential constraints exist and ENFORCE(CONSTRAINTS) is specified.
[4] If tape is specified, the maximum key length of all indexes involved in the sort phase must be a minimum of 6 bytes. This length, when added to the internally assigned 12-byte header, must be at least 18 bytes as required by DFSORT.
[5] Required for LOAD with COPYDDN or RECOVERYDDN and for REORG with COPYDDN, RECOVERYDDN, SHRLEVEL REFERENCE, or SHRLEVEL CHANGE.
[6] Required unless you specify NOSYSREC or SHRLEVEL CHANGE.
[7] Data sets can not be shared between SORTOUT and SYSUT1. Sharing these data sets can cause unpredictable results.
[8] If you specify FILTERDDN, there is no default DD name. You must supply a name.

For input data sets, the online utilities use the logical record length (LRECL), record format (RECFM), and 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 SQL/DS™ data in VBS format.

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

For both input and output data sets, the online utilities use the value 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 used to 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.

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

Concatenating data sets

DB2 utilities let you concatenate unlike input data sets. Therefore, the data sets in a concatenation list can have differing 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 eight 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 they are no longer needed 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 the contents are no longer needed for subsequent loads.
- Retain the discard data set until the contents are no longer needed for subsequent loads.

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

- Use DISP=(MOD,CATLG,CATLG) or DISP=(MOD,CATLG) for data sets 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 SORTWKnn data sets, or refer to *DFSORT Application Programming: Guide* for alternatives.
- Do not use temporary data set names.

Refer to Table 84 on page 518 and Table 85 on page 519 for information about the default data dispositions specified for dynamically-allocated data sets.

Security

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

Using the DB2 utilities panel in DB2I

Using the DB2 Utilities panel to execute DB2 online utilities requires the least knowledge of JCL.

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

Editing and submitting a utility job: 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 procedure outlined in the following example:

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

   For example, the following utility control statement makes an incremental image copy of table space DSN8D61A.DSN8S61D with a SHRLEVEL value of CHANGE:

   ```
   COPY TABLESPACE DSN8D61A.DSN8S61D
     FULL NO
     SHRLEVEL CHANGE
   ```

   For the rest of this example, suppose that you save the statement in the default data set, UTIL.

2. Select the DB2I menu from the ISPF Primary Option Menu.
3. Select the UTILITIES option on the DB2I Utilities panel. Items you must specify are highlighted on the DB2 Utilities panel 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.)

  4 CONTROL CARDS DATA SET    ===> UTIL

 To RESTART a utility, specify starting point, otherwise specify NO.

  5 RESTART  ===> NO         (NO, At CURRENT position, or beginning of PHASE)

  *  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 you want to execute. In this example, you want to submit the utility job, but you want to edit the JCL first. After you have edited the utility job, specify `SUBMIT` on the editor command line.
5. Field 2 must be a unique identifier for your utility job. The default value TEMP. In this example, that value is satisfactory; leave it as is.
6. Fill in field 3 with the utility you want to run. Specify REORG LOB to indicate REORG TABLESPACE of a LOB table space.

   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 only if this job restarts a stopped utility. In this example, leave the default value, NO.
9. Press ENTER.

If you specify COPY, LOAD, MERGECOPY, or REORG TABLESPACE as the utility in field 3, you must fill in the appropriate fields on the “Data Set Names” panel. In this example, COPY was specified.

```
DSNEUP02                          DATA SET NAMES
 ===>

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

   Enter data set name for
LOAD or REORG TABLESPACE:
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:
7 PUNCHDSN ==>
 PRESS:  ENTER to process     END to exit    HELP for more information
```

Figure 2. Data set names panel

1. Fill in field 1 if you are running LOAD or REORG. If you are running LOAD, you must specify the data set name that contains records to be loaded. If you are running REORG you must specify the unload data set. In this example, you do not have to fill in field 1.
2. Fill in field 2 if you are running LOAD or REORG with discard processing; you must specify a discard data set. In this example, you do not have to fill in field 2.

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 the current site if you are running MERGECOPY. The DDNAME generated by the panel for this field is SYSCOPY. This is an optional field for LOAD and for REORG with SHRLEVEL NONE; it is required for COPY, for MERGECOPY, and for REORG with SHRLEVEL REFERENCE or CHANGE.
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 *DDNAME* generated by the panel for this field is SYSCOPY2. This is an optional field. In this example, you do not have 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 *DDNAME* generated by the panel for this field is SYSRCOPY1. This is an optional field.
6. Fill in field 6 with the backup output data set for the recovery site if you are running COPY, LOAD, or REORG. The *DDNAME* generated by the panel for this field is SYSRCOPY2. This field is optional. In this example, you do not have 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 or REORG DISCARD. The *DDNAME* generated by the panel for this field is SYSPUNCH. In this example, you do not have to fill in field 7.
8. Press ENTER.

If you need help while using the DB2 Utilities panel or the Data Set Names panel, press the HELP PF key. HELP panels explain the parameters on the DB2 Utilities panel and show the syntax and some sample utility control statements for each online utility.

Using the DSNU CLIST command in TSO

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

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

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

To use the DSNU CLIST command:

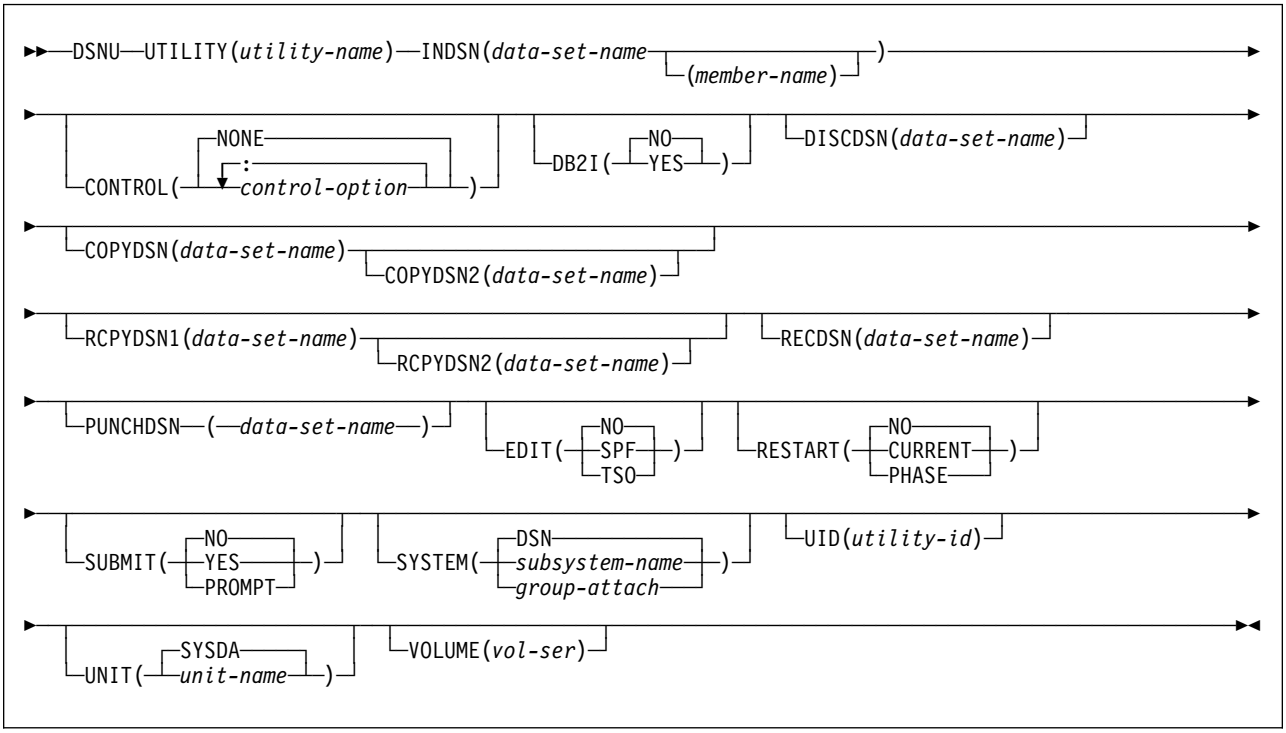
1. Create a file containing the required utility statements and control statements. The file is used to create the SYSIN data set in the generated job stream. Do not include double-byte character set (DBCS) data in this file.
2. Ensure that the DB2 CLIST library is allocated to the *ddname* SYSPROC.

3. Execute the command procedure by using the syntax in "DSNU CLIST command syntax" on page 35.
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 40 explains how the edit the JCL data set.

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

DSNU CLIST command syntax



DSNU CLIST option descriptions

The parentheses 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 you want to execute. Select the name from the following list:

CHECK DATA	MERGE	REORG LOB
CHECK INDEX	MODIFY	REORG TABLESPACE
CHECK LOB	QUIESCE	REPAIR
COPY	REBUILD	REPORT
DIAGNOSE	RECOVER	RUNSTATS
LOAD	REORG INDEX	STOSPACE

DB2 places the JCL in a data set named DSNU*xxx*.CNTL, where DSNU*xxx* is a control file name. The control file contains the statements 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 the **UID** keyword on page 38 for a list of the online utilities and the control file name associated with each utility.

INDSN(*data-set-name* **(***member-name***))**
Specifies what data set 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 **CONTROL**(**NONE**).

control-option Lists one or more of the options given below. 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 **DB2I**(**NO**).

(YES) Indicates that DSNU CLIST command is called from the DB2I environment and that card information can be used. Only the utility panels should execute the CLIST command with DB2I(YES).

DISCDSN(*data-set-name***)**
The cataloged data set name used by LOAD and REORG as a discard data set. For LOAD, this data set holds records not loaded; for REORG, it holds records not reloaded.

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

COPYDSN(*data-set-name***)**
The name of the cataloged data set that is used as a target (output) data set. If you do not supply this information, the CLIST command prompts you for it. It 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 is used as a target (output) data set for the backup copy. It is optional for COPY, MERGECOPY, LOAD, and REORG.

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

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

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

EDIT Specifies whether to invoke an editor to edit the temporary file generated by the CLIST command.

(NO) Does not invoke an editor.

The **default** is **EDIT**(**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 **RESTART**(**NO**).

(CURRENT) Restarts the utility at the last commit point.

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

SUBMIT Specifies whether to submit the generated JCL for processing.

(NO) Does not submit the JCL data set for processing.

The **default** is **SUBMIT**(**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 **SYSTEM** (**DSN**).

UID(*utility-id***)**

Provides a unique identifier for this utility job within DB2.

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

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
REBUILD INDEX	DSNUREB
RECOVER	DSNUREC
REORG INDEX	DSNURGI
REORG LOB	DSNURGL
REORG TABLESPACE	DSNURGT
REPAIR	DSNUREP
REPORT	DSNURPT

RUNSTATS	DSNURUN
STOSPACE	DSNUSTO

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. *unit-name* is placed after the UNIT clause of the generated DD statement.

The **default** is **UNIT**(**SYSDA**).

VOLUME(*vol-ser***)**

Assigns the serial number of the volume on which a new temporary or permanent data set resides. *vol-ser* is placed 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.

Reviewing 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 3. Any of these statements can be edited.

```
//DSNUCOP JOB your-job-statement-parameters
//          USER=userid,PASSWORD=userword
//*ROUTE PRINT routing-information
//UTIL      EXEC  DSNUPROC,SYSTEM=DSN,UID=TEMP,UTPROC='
//SYSCOPY   DD    DSN=MYCOPIES.DSN8D61A.JAN1,DISP=(MOD,CATLG,CATLG),
//          UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN     DD    *
     COPY TABLESPACE DSN8D61A.DSN8S61D
          FULL NO
          SHRLEVEL CHANGE

/*
```

Figure 3. 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 after the JCL is complete. The job name is DSNU, followed by the first three letters of the utility name you are using.
EXEC	The CLIST command builds the EXEC statement. The values 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 used by online utilities” on page 28.

SYSPRINT DD SYSOUT=A

Utility messages are sent to the SYSPRINT data set. The generated JCL defines OUTPUT, SYSPRINT as SYSOUT=A. 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 further information, see *OS/390 TSO/E Command Reference.*

UTPRINT DD SYSOUT=A

If any utility requires a sort, it executes DFSORT. Messages from that program are sent to UTPRINT. The generated JCL defines UTPRINT as SYSOUT=A.

SYSIN DD *

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

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, change JCL parameters (such as *ddnames*), or change utility control statements.

If you use a *ddname* that is not the default on some utility statement that you use, you must change the *ddname* in the JCL 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 called *authorization-id*.DSNURGT.CNTL 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 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 and will not be 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.

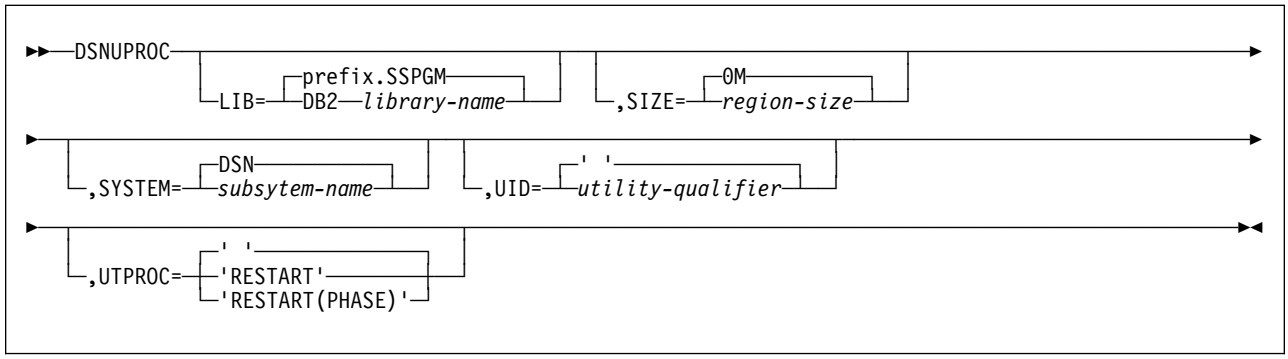
```
%DSNU
  UTILITY (COPY)
  INDSN ('MYCOPY(STATEMNT)')
  COPYDSN ('MYCOPIES.DSN8D61A.JAN1')
  EDIT (TSO)
  SUBMIT (YES)
  UID (TEMP)
  RESTART (NO)
```

Using the supplied JCL procedure (DSNUPROC)

Another method of invoking a DB2 online utility uses the supplied JCL procedure, DSNUPROC, shown in Figure 4 on page 42. This procedure uses the parameters that you supply to build an appropriate EXEC statement that executes an online utility.

To execute the DSNUPROC procedure, you must write and submit a JCL data set like that built by the DSNU CLIST command, and shown in Figure 3 on page 39. In your JCL, the EXEC statement executes the DSNUPROC procedure. "DSNUPROC syntax" explains the parameters you can supply to that procedure and the syntax.

DSNUPROC syntax



DSNUPROC option descriptions

The following list describes all the parameters. For example, in Figure 3 on page 39, you need to use only one parameter, UID=TEMP; for all others, you can use the defaults.

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 number of bytes of virtual storage 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').

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 last commit point
'RESTART(PHASE)'	To restart at the beginning of the phase executed last.

The 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 used by online utilities” on page 28 for a description of data sets that you might need.

Figure 4 is the DSNUPROC procedure that was executed by the JCL example in Figure 3 on page 39.

Sample DSNUPROC listing

```
//DSNUPROC PROC LIB='prefix.SSPGM',
//         SYSTEM=DSN,
//         SIZE=0K,UID=',UTPROC='
//*
//*********************************************************************
//*                                                                   *
//* PROCEDURE-NAME:      DSNUPROC                                     *
//*                                                                   *
//* DESCRIPTIVE-NAME:    UTILITY PROCEDURE                            *
//*                                                                   *
//*   COPYRIGHT = 5665-DB2 (C) COPYRIGHT IBM CORP 1982, 1993          *
//*   REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083           *
//*                                                                   *
//* STATUS:              Version 6                                       *
//*                                                                   *
//* FUNCTION:  THIS PROCEDURE INVOKES THE ADMF UTILITIES IN THE       *
//*             BATCH ENVIRONMENT                                     *
//*                                                                   *
//* PROCEDURE-OWNER:     UTILITY COMPONENT                            *
//*                                                                   *
//* COMPONENT-INVOKED:   DB2 UTILITIES (ENTRY POINT DSNUTILB).        *
//*                                                                   *
//* ENVIRONMENT:         TSO BATCH                                    *
//*                                                                   *
//* INPUT:                                                            *
//*      PARAMETERS:                                                  *
//*           LIB    = THE DATA SET NAME OF THE DB2  PROGRAM LIBRARY.  *
//*                    THE DEFAULT LIBRARY NAME IS PREFIX.SSPGM,    *
//*                    WITH PREFIX SET DURING INSTALLATION.           *
```

Figure 4 (Part 1 of 2). Sample listing of supplied JCL procedure DSNUPROC

```
//*           SIZE   = THE REGION SIZE OF THE UTILITIES EXECUTION AREA.*
//*                    THE DEFAULT REGION SIZE IS 4096K.               *
//*           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;
//*
//********************************************************************
//*  DATA SETS USED BY THE UTILITY                                   *
//********************************************************************
//*
//SYSPRINT DD   SYSOUT=*
//UTPRINT  DD   SYSOUT=*
//SYSUDUMP DD   SYSOUT=*
//*DSNUPROC PEND        REMOVE * FOR USE AS INSTREAM PROCEDURE
```

Figure 4 (Part 2 of 2). Sample listing of supplied JCL procedure DSNUPROC

Creating the JCL data set yourself

DB2 online utilities execute as standard OS/390 jobs. To execute the utility, you must supply the JOB statement required by your installation and the JOBLIB or STEPLIB DD statements required to access DB2. You must also include an EXEC statement and a set of DD statements. The EXEC statement is described in “Data sets used by online utilities” on page 28. For a description of the DD statements you might need, see “Data sets used by online utilities” on page 28.

We recommend using 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 41.

EXEC statement

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

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

where 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.

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 last commit point
`'RESTART(PHASE)'`	To restart at the beginning of the phase executed last.

For the example in Figure 4 on page 42, you write:

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

Chapter 2-2. Monitoring and controlling online utilities

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

Monitoring utilities with the DISPLAY UTILITY command

The information under this heading, up to "Running utilities concurrently" on page 46 is General-use Programming Interface and Associated Guidance Information, as defined in Appendix E, "Notices" on page 545.

Use the DB2 DISPLAY UTILITY command to check the current status of online utilities. Figure 5 on page 46 shows an example of the output generated by the DISPLAY UTILITY command. In the example output, DB2 returns a message that indicates the member name (**A**), utility name (**B**), identifier (**C**), phase (**D**), and status (**E**). The message also indicates the number of pages or records that are processed by the utility (**F**)[1]. The output might also report additional information for an executing utility, such as log phase estimates or utility subtask activity.

Determining the status of a utility

An online utility can have one of these statuses:

Status (**E**)	Description
Active	The utility has started execution.
Stopped	The utility has abnormally stopped executing before completion, but the table spaces and indexes accessed by the utility remain under utility control. To make the data available again, you must either: • Correct the condition that stopped the utility, and restart the utility so that it runs to termination, or • Terminate the utility with the DB2 TERM UTILITY command (see "Terminating an online utility with the TERM UTILITY command" on page 47).
Terminated	The utility has been requested to terminate by the DB2 TERM UTILITY command. If the utility has terminated, there is no message.

Determining which utility phase is currently executing

DB2 online utility execution is divided into phases. Each utility starts with the UTILINIT phase that performs initialization and set up. Each utility finishes with a UTILTERM phase that cleans up after processing has completed. The other phases of online utility execution differ, depending on the utility. See the

[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 issued from a different member, the count may lag substantially.

```
DSNU100I - DSNUGDIS - USERID = SAMPID
            A  MEMBER = DB1G
            C  UTILID = RUNTS
               PROCESSING UTILITY STATEMENT 1
            B  UTILITY = RUNSTATS
            D  PHASE = RUNSTATS   F  COUNT = 0
            E  STATUS = STOPPED
DSN9022I - DSNUGCC  '-DISPLAY UTILITY' NORMAL COMPLETION
```

Figure 5. DISPLAY UTILITY command sample output

"Execution Phases" section in the descriptions of each utility. Output from the DISPLAY UTILITY command shows the phase that is currently executing.

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.

During execution of the utility, any of these problems can cause a failure:

- **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 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, terminate the job step (by using TERM UTILITY (uid)) and resubmit it.

- **Problem:** DB2 terminates the utility and issues return code 8.

  Solution: One or more objects may be in a restrictive or advisory status. See Appendix C, "Resetting an advisory or restrictive status" on page 527 for more information on resetting the status of an object.

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

Running utilities concurrently

Some online utilities allow other utilities and SQL statements to run concurrently on the same target object. The online utility descriptions in Section 2 of this book feature a section on compatibility and concurrency. The section on concurrency and compatibility 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 is also shown.

See Section 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 utilities: 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 MVS system where the specified DB2 subsystem is running. You must be sure that MVS runs the utility job on the appropriate MVS system. You must use one of several MVS installation-specific statements to make sure this happens. These include:

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

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

- For JES3 systems, the following statement can be inserted into the utility JCL:

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

Those JCL statements are described in *OS/390 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 (with 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 and then you can terminate the utility from any system.

You can only restart a utility 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 MVS 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 48 is General-use Programming Interface and Associated Guidance Information, as defined in Appendix E, “Notices” on page 545.

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

After you issue the TERM UTILITY command, you cannot restart the terminated utility. It is possible that 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 are particularly important when terminating the COPY, LOAD, and REORG utilities.

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

In a data sharing environment, TERM UTILITY is effective for active utilities when 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.

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 6 shows a sample job stream for our COPY example:

```
//TERM EXEC PGM=IKJEFT01,COND=((8,GT,S1),EVEN)
//*
//*************************************************************
//* IF THE PREVIOUS UTILITY STEP, S1, ABENDS, ISSUE A
//* TERMINATE COMMAND.  IT CAN NOT 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 6. 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, it is often possible to restart it. With the restart procedure, you avoid repeating much of the work that had already been done.

Two different methods of restart are available:

Phase restart can be done from the beginning of the phase that was being processed.

Current restart can be done from the last checkpoint 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.

Updating the JCL data set for restarting a utility

To restart a DB2 online utility, update the original JCL data set with the RESTART parameter. This can be accomplished using one of three methods:

- **Using DB2I**. Restart the utility following these steps:
  1. Access the DB2 UTILITIES panel.
  2. Fill in the panel fields as documented in Figure 2 on page 33, except for field 5.
  3. Change field 5 to CURRENT or PHASE, depending on the method of restart desired.
  4. Press ENTER.
- **Using the DSNU CLIST command**. Restart the utility by invoking the DSNU CLIST command as described in "Using the DSNU CLIST command in TSO" on page 34, but change the value of the RESTART parameter, using either RESTART or RESTART(PHASE).
- **Creating your own JCL**. If you create your own JCL you must specify RESTART or RESTART(PHASE) to restart the utility. You must also check the DISP parameters on the DD statements. For example, DD statements that have DISP=NEW and need to be reused must have DISP changed to OLD or MOD. If generation data groups (GDGs) are used and any (+1) generations were cataloged, then 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.

Adding or deleting utility statements

Restart processing remembers the relative position of the 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.

Restarting after the output data set is full

Special considerations exist when restarting a utility at the last commit point after the output data set runs out of space (for example, ABENDx37 on SYSUT1).

If you receive an out-of-space condition on the output data set, follow these steps before restarting the utility at the last commit point:

1. Copy the output data set to a temporary data set. Use the same DCB parameters. Use MVS utilities that do not reblock the data set during the copy operation (for example, DFDSS ADRDSUU or DFSORT ICEGENER). Avoid using the MVS utilities IEBGENER or ISPF 3.3.
2. Delete or rename the output data set. 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. You should know the current DCB parameters before attempting to redefine the data set.
3. Copy the data from the temporary data set into the new, larger output data set. Use MVS utilities that do not reblock the data set during the copy operation (for example, DFDSS ADRDSUU or DFSORT ICEGENER).

Other restart hints

The following guidelines provide additional information for restarting utilities:

- The VOLSER (if the data set is not cataloged) and the data set name are used to track utility restart information. They must be the same as what you specified in the original JCL data set for the utility you want to restart.
- When restarting a utility with cataloged data sets, do not specify VOL=SER=. Let DB2 determine the VOLSER of the data sets from the system catalog.
- 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 subsequent utility steps.
- Do not specify MVS automatic step restart.
- If a utility is restarted in the UTILINIT phase, it is re-executed from the beginning of the phase.
- When you restart a LOAD, REBUILD INDEX, or REORG job in which you specified the STATISTICS keyword, DB2 does not collect inline statistics. You should run the RUNSTATS utility after the restarted utility completes.
- If a utility abends or system failure occurs while the utility is in the UTILTERM phase, you must restart it with RESTART(PHASE).

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

Chapter 2-3. CATMAINT

The CATMAINT utility updates the catalog; it should be run only during migration to a new release of DB2.

Output: Output for CATMAINT UPDATE is the updated catalog.

Authorization required: The authorization required for CATMAINT is installation SYSADM.

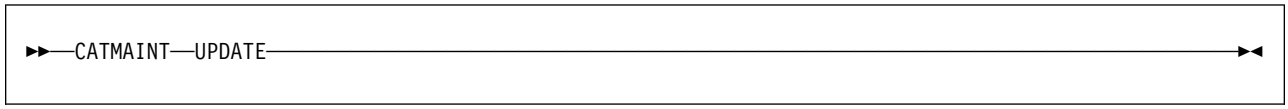
Execution phases of CATMAINT: The CATMAINT utility operates in these phases:

Phase	Description
UTILINIT	Initialization and setup to the new type
UTILTERM	Cleanup

Syntax and options of the 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, 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

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see “Control statement coding rules” on page 27.

UPDATE Indicates you want to update the catalog. Run this option only when you migrate to a new release of DB2.

Instructions for running CATMAINT

To run CATMAINT, you must:

1. Read “Before running CATMAINT” on page 52.
2. Prepare the necessary data sets, as described in “Data sets used by CATMAINT” on page 52.
3. Create JCL statements by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27.

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 52. (For a complete description of the syntax and options for CATMAINT, see "Syntax and options of the control statement.")
5. Check "Concurrency and compatibility" on page 53 if you want to run other jobs concurrently on the same target objects.
6. Plan for restarting CATMAINT if the job doesn't complete, as described in "Terminating or restarting CATMAINT" on page 53.
7. Run CATMAINT.

See "Chapter 2-1. Invoking DB2 online utilities" on page 27 for an explanation of methods you can use to execute DB2 utilities.

Before running CATMAINT

The work file database is used for CATMAINT sorting. Prior to executing the CATMAINT utility, calculate the size of the work file database.

To calculate the size of the work file database, see *DB2 Installation Guide*.

Data sets used by CATMAINT

Table 2 describes the data sets used by CATMAINT. Include DD statements in all data sets that are used by your job.

Table 2. Data sets used by CATMAINT

Data Set	Description	Required?
SYSIN	An input data set containing the utility control statement	Yes
SYSPRINT	An output data set for messages	Yes

Instructions for specific tasks

To perform the following task, specify the options and values for that task in your utility control statement.

Updating the catalog for a new release: When you migrate to a new release of DB2, you must update the catalog for the prior release to the new version. The DSNTIJTC job, described in Section 2 (Volume 1) of *DB2 Installation Guide*, runs CATMAINT UPDATE to update the catalog. Run CATMAINT UPDATE only when you migrate to a new release. DB2 displays message DSNU777I at several points during CATMAINT execution.

If you still have type 1 indexes, shared read-only data, or data set passwords, the CATMAINT UPDATE utility abnormally terminates. See "Migrating the DB2 Subsystem" in Section 2 of *DB2 Installation Guide* for the steps necessary to migrate to a new release.

If necessary, message DSNU776I or DSNU778I can give you information about why an abend occurred.

Terminating or restarting CATMAINT

You can terminate CATMAINT using the TERM UTILITY command, but it leaves the indexes in REBUILD pending status. See “Resetting the REBUILD pending status” on page 175 for information on resetting this status.

You cannot restart CATMAINT.

Concurrency and compatibility

Catalog and directory index availability: The catalog or directory indexes are not available while CATMAINT is running. This can cause other jobs to time out with message DSNT376I or message DSNT501I.

Chapter 2-4. CHECK DATA

The CHECK DATA online utility checks table spaces for violations of referential and table check constraints, and reports information about violations that are detected.

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. CHECK DATA can be run against a base table space only, not a LOB table space.

For a diagram of CHECK DATA syntax and a description of available options, see “Syntax and options of the control statement” on page 56. For detailed guidance on running this utility, see “Instructions for running CHECK DATA” on page 59.

Output: CHECK DATA optionally deletes rows that violate referential or table check constraints. A row that violates one or more constraints is copied, once, to an exception table.

If any violation of constraints is found, CHECK DATA puts the table space being checked in the CHECK pending status.

On successful execution, CHECK DATA resets the CHECK pending status.

Authorization required: To run this utility, the privilege set of this process must include one of the following:

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

Execution phases of CHECK DATA:

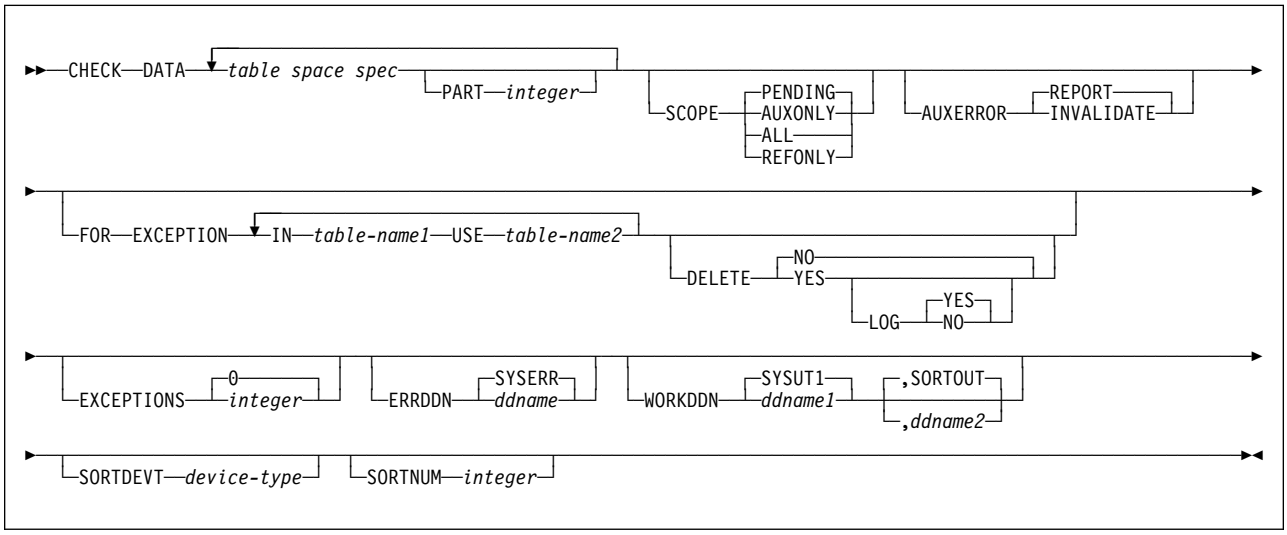
Phase	Description
UTILINIT	Initialization
SCANTAB	Extract foreign keys; use foreign key index if it exists, else scan table
SORT	Sort foreign keys if not extracted from foreign key index
CHECKDAT	Look in primary indexes for foreign key parents, and issue messages to report errors detected
REPORTCK	Copy error rows into exception tables, and delete them from source table if DELETE YES is specified
UTILTERM	Cleanup

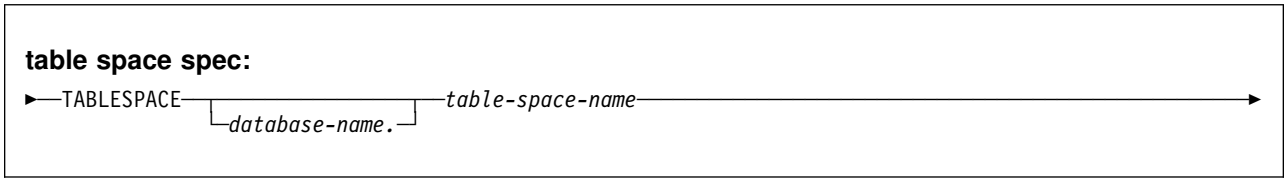
Syntax and options of the control statement

The utility control statement defines the function 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

For guidance in interpreting syntax diagrams, see "How to read the syntax diagrams" on page 4.





Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see "Control statement coding rules" on page 27.

DATA Indicates that you are checking referential and table check 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 defined for the table space. The maximum is 254.

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

PENDING Indicates that only those 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 table space is ignored.

The **default** is **SCOPE 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 the LOB column check is not performed.

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

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 **AUXERROR REPORT.**

INVALIDATE A LOB column check error is reported with a warning message. The base table LOB column is set to an invalid status. The base table space is set to the auxiliary warning (AUXW) 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 in violation of referential or table check constraints is copied to an exception table.

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

IN *table-name1*
Specifies the table (in the table space specified on the TABLESPACE keyword) from which rows are to be copied.

table-name1 is the name of the table.

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.

DELETE Indicates whether or not rows in violation of referential or table check constraints are deleted from the table space. You can only use this option if you have used the FOR EXCEPTION keyword.

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

The **default** is **DELETE NO**.

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

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

LOG Specifies the logging action taken when records are deleted.

YES Logs all records 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 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. You cannot recover data across a point in the log in which CHECK DATA DELETE YES LOG NO was used.

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

Only records containing *primary* referential integrity errors or table check constraint violations are applied toward the exception limit. There is no limit on the number of records containing secondary errors.

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

ERRDDN *ddname*

Specifies a DD statement for an error processing data set.

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

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 to be dynamically allocated by DFSORT. It can be any device type acceptable to the DYNALLOC parameter of the SORT or OPTION control statement for DFSORT, as described in *DFSORT Application Programming: Guide.*

device-type is the device type. If you omit SORTDEVT and a sort is required, you must provide the DD statements that the sort program requires for the temporary data sets.

SORTNUM *integer*

Tells the number of temporary data sets to be dynamically allocated by the sort program.

integer is the number of temporary data sets.

If you omit SORTDEVT, SORTNUM is ignored.

If you use SORTDEVT and omit SORTNUM, no value is passed to DFSORT; it is allowed to take its own default.

Instructions for running CHECK DATA

To run CHECK DATA, you must:

1. Read “Before running CHECK DATA” on page 60.
2. Prepare the necessary data sets, as described in “Data sets used by CHECK DATA” on page 63.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27. (For examples of JCL for CHECK DATA, see “Sample control statements” on page 68.)
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 64. (For a complete description of the syntax and options for CHECK DATA, see “Syntax diagram” on page 56.)

5. Check the compatibility table in “Concurrency and compatibility” on page 67 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 66.
7. Run CHECK DATA.

See “Chapter 2-1. Invoking DB2 online utilities” on page 27 for a description of ways to execute DB2 utilities.

Before running CHECK DATA

For a table with no LOB columns:

You should run CHECK INDEX on primary key indexes and foreign key indexes before running CHECK DATA to ensure that the indexes used by CHECK DATA are valid. This is especially important before using CHECK DATA with DELETE YES.

For a table with LOB columns:

If you plan to run CHECK DATA on a base table space containing 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 prior to running CHECK DATA to ensure the validity of the LOB table space and index on the auxiliary table.
3. Run CHECK INDEX on the base table space indexes.

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 will issue an error message and fail.

Create exception tables:

An *exception table* is a user-created table that duplicates the definition of a dependent table. The dependent table is checked with the CHECK DATA utility. It 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 3 describes the contents of an exception table.

Table 3. Exception tables

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

Notes:

1. You can use CHAR(5) for any type of table space.

If you delete rows using the CHECK DATA utility, 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 will be deleted.

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

- The exception tables should not have any unique indexes or referential or table check constraints that might cause errors when CHECK DATA inserts rows in them.
- You can create a new exception table before you run CHECK DATA, or 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 this column.
- You must have DELETE authorization on the dependent table being checked.
- You must have INSERT authorization on the exception table.
- Column names in the exception table can be given 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, and 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 a LOB column check error only does not participate in exception processing.

Example: creating an exception table for the project activity table

General-use Programming Interface

There is a clause of CREATE TABLE that makes the exception table easy to create. You can create an exception table for the project activity table by using these SQL statements:

```
CREATE TABLE EPROJACT
   LIKE DSN8610.PROJACT
   IN DATABASE DSN8D61A;

ALTER TABLE EPROJACT
   ADD RID CHAR(4);

ALTER TABLE EPROJACT
   ADD TIME TIMESTAMP NOT NULL WITH DEFAULT;
```

The first statement requires the SELECT privilege on table DSN8610.PROJACT and the privileges usually required to create a table.

Table EPROJACT has the same structure as table DSN8610.PROJACT, but it has two extra columns:

- 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 have to be. However, the rest of the column attributes for the initial columns must be same as those of the table being checked.
- The next column, 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, you have to use a different name. The description of this column, CHAR(4), is required.
- The final timestamp column is also useful. If the timestamp column is defined, 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 detected by CHECK DATA. The TIME column allows you to identify rows that were added by the most recent run.

Eventually, you correct the data in the exception tables, perhaps with an SQL UPDATE, and transfer the corrections to the original tables by using statements such as:

```
INSERT INTO DSN8610.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 will fail and issue error message DSNU075E.

Data sets used by CHECK DATA

Table 4 describes the data sets used by CHECK DATA. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 4. Data sets used by CHECK DATA

Data Set	Description	Required?
SYSIN	An input data set containing 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. The symbolic names of the DD statement are specified with 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 page 64.	Yes
Error data set	An output data set that collects information about violations encountered during the CHECKDAT phase for referential constraints or the SCANTAB phase for check constraints. The symbolic name of the DD statement is specified with the ERRDDN parameter of the utility control statement. The default *ddname* is SYSERR.	Yes
UTPRINT	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 to be checked. It is named in the CHECK DATA control statement and is accessed through the DB2 catalog. (If you want to check only one partition of a table space, use the PART option in the control statement.)

Exception tables

For each table in a table space that is checked, you can specify the name of an exception table in the utility control statement. Any row that violates a referential constraint is copied to the associated exception table. See page 60 for more information.

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

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, then go to step 3. If a table space does not have a LOB column, then go to step 2.
2. Add 9 to the length of the longest foreign key.
3. Multiply the sum by the number of keys and LOB columns checked.

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

Creating the control statement

See "Syntax diagram" on page 56 for CHECK DATA syntax and option descriptions. See "Sample control statements" on page 68 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:

Specify the scope of CHECK DATA

To specify the scope of CHECK DATA, it is normally sufficient to run CHECK DATA with SCOPE PENDING. DB2 keeps track of the data rows that must be checked to ensure the referential integrity of the table space. You should run SCOPE ALL whenever the scope information is in doubt. The scope information is recorded in the DB2 catalog. The scope information can become in doubt whenever you start the target table space with ACCESS(FORCE), or the catalog is recovered to a point in time.

If you only want to check 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.

Finding violations

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

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

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 nonreferential 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.
- Deleting a row might make the time error harder to detect.
- Valid rows might be detected.

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

Resetting CHECK pending status

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.

Take one of the following actions to remove the 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 in violation of referential or table check constraints.

Interpreting LOB column errors

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

Orphan LOBs: An orphan LOB column is a LOB found in the LOB table space but not referenced by the base table space. An orphan can result if you recover the base table space to a point in time prior to the insertion of the base table row or prior to the definition of the LOB column. An orphan can also result if you recover the LOB table space to a point in time prior to the deletion of a base table row.

Missing LOBs: A missing LOB column is a LOB referenced by the base table space, but the LOB is not in the LOB table space. A missing LOB can result if you recover the LOB table space to a point in time when the LOB column is not in the LOB table space. This could be a point in time prior to the first insertion of the LOB into the base table, or when the LOB column is null or has a zero length.

Out-of-synch LOBs: An out-of-synch LOB error occurs when DB2 detects 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. An LOB column is also out-of-synch if the base table LOB column 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 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 reports a message identifying the table, row, column, and type of error. Any additional actions depend on the option you specify for the AUXERROR parameter.

Actions performed with AUXERROR REPORT: DB2 sets the base table space to the CHECK pending 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 status.

Actions performed with AUXERROR INVALIDATE: DB2 sets the base table LOB column to an invalid status, and sets the base table space to the auxiliary warning (AUXW) status. You can use SQL to update a LOB column in the AUXW status, however, any other attempt to access the column will result in a -904 SQL return code.

See Appendix C, “Resetting an advisory or restrictive status” on page 527 for information about the resetting the restrictive table space status.

Terminating or restarting CHECK DATA

Terminating CHECK DATA

When you terminate CHECK DATA, table spaces remain in the CHECK pending status as they were when the utility was terminated. The CHECKDAT phase places the table space in the CHECK pending status when an error is detected; at the end of the phase, the CHECK pending status is reset if no errors were detected. 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 47.

Restarting CHECK DATA

You can restart a CHECK DATA utility job, but it starts from the beginning again.

For instructions on restarting a utility job, see “Restarting an online utility” on page 48.

Concurrency and compatibility

Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions of the same table space or index space are compatible.

Claims and drains: Table 5 shows the claim classes in which CHECK DATA claims and drains and any restrictive status the utility sets on the target object.

Table 5. Claim classes of CHECK DATA operations. Use of claims and drains; restrictive state set on the target object.

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
Nonpartitioning index	DW/UTRO	DA/UTUT		DR
Logical partition of index			DW/UTRO	DA/UTUT
Primary index	DW/UTRO	DW/UTRO	DW/UTRO	DW/UTRO
RI dependent and descendent table spaces and indexes		DA/UTUT		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
- Blank: Object is not affected by this utility

Table 6 (Page 1 of 2). Claim classes on a LOB table space and index on the auxiliary table for CHECK DATA operations. Use of claims and drains; restrictive states set on the target object.

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

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

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 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 control statements

Example 1: CHECK DATA with DELETE. The following shows CHECK DATA JCL for checking and deleting.

```
//STEP1    EXEC DSNUPROC,UID='IUIQU1UQ.CHK1',
//         UTPROC='',
//         SYSTEM='V61A'
//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 DSN8D61A.DSN8S61D
           TABLESPACE DSN8D61A.DSN8S61E
     FOR EXCEPTION IN DSN8610.DEPT        USE DSN8610.EDEPT
                   IN DSN8610.EMP         USE DSN8610.EEMP
                   IN DSN8610.PROJ        USE DSN8610.EPROJ
                   IN DSN8610.PROJACT     USE DSN8610.EPROJACT
                   IN DSN8610.EMPPROJACT  USE DSN8610.EEPA
     DELETE YES
//*
```

Example 2: Control statement for deleting error rows. Check for and delete all constraint violations in table spaces DSN8D61A.DSN8S61D and DSN8D61A.DSN8S61E.

```
CHECK DATA TABLESPACE DSN8D61A.DSN8S61D
 TABLESPACE DSN8D61A.DSN8S61E
 FOR EXCEPTION IN DSN8610.DEPT USE DSN8610.EDEPT
               IN DSN8610.EMP USE DSN8610.EEMP
               IN DSN8610.PROJ USE DSN8610.EPROJ
               IN DSN8610.PROJECT USE DSN8610.EPROJECT
               IN DSN8610.EMPPROJECT USE DSN8610.EEMPPROJECT
 DELETE YES
```

Chapter 2-5. CHECK INDEX

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

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

It should also be used before CHECK DATA to ensure that the indexes used by CHECK DATA are valid. This is especially important before using CHECK DATA with DELETE YES. 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 a diagram of CHECK INDEX syntax and a description of available options, see “Syntax and options of the control statement” on page 72. For detailed guidance on running this utility, see “Instructions for running CHECK INDEX” on page 73.

Output: CHECK INDEX generates several messages that show whether the indexes are consistent with the data. See Section 3 of *DB2 Messages and Codes* for more information about these messages.

For unique indexes, any two null values are taken 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, and CHECK INDEX does not issue an error message.

CHECK INDEX issues an error message if there are 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, the privilege set of this process must include one of the following:

- 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.

Execution Phases of CHECK INDEX:

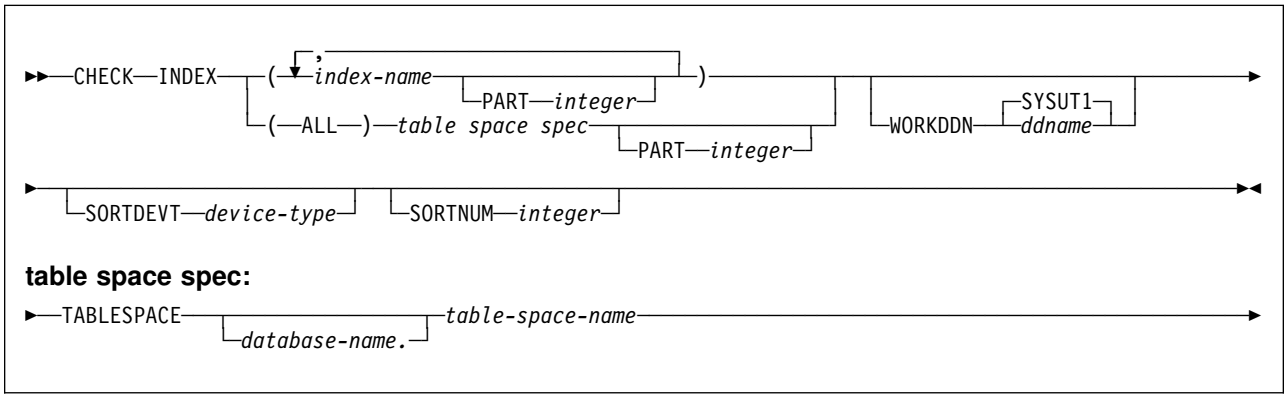
Phase	Description
UTILINIT	Initialization and setup
UNLOAD	Unloading of index entries
SORT	Sorting of unloaded index entries
CHECKIDX	Scanning of data to validate index entries
UTILTERM	Cleanup.

Syntax and options of the control statement

The utility control statement defines the function 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

For guidance in interpreting syntax diagrams, see "How to read the syntax diagrams" on page 4.



Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see "Control statement coding rules" on page 27.

INDEX Indicates that you are checking for index consistency.

(*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 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.

PART *integer* Identifies a physical partition of a partitioning index or a logical partition of a nonpartitioning index to check.

integer is the number of the partition and must be in the range from 1 to the number of partitions defined for the table space. The maximum is 254.

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

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

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

WORKDDN *ddname*

Specifies a DD statement for a temporary work file.

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

SORTDEVT *device-type*

Specifies the device type for temporary data sets to be dynamically allocated by DFSORT. It can be any device type acceptable to the DYNALLOC parameter of the SORT or OPTION control statement for DFSORT.

device-type is the device type. If you omit SORTDEVT and a sort is required, you must provide the DD statements that the sort program requires for the temporary data sets.

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

integer is the number of temporary data sets.

If you omit SORTDEVT, SORTNUM is ignored.

If you use SORTDEVT and omit SORTNUM, no value is passed to DFSORT; it is allowed to take its own default.

Instructions for running CHECK INDEX

To run CHECK INDEX, you must:

1. Prepare the necessary data sets, as described in 71.
2. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27. (For examples of JCL for CHECK INDEX, see “Sample control statements” on page 77.)
3. Prepare a utility control statement, specifying the options for the tasks you want to perform. For a complete description of the syntax and options for CHECK INDEX, see “Syntax and options of the control statement” on page 72.
4. Check the compatibility table in “Concurrency and compatibility” on page 76 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 76.
6. Run CHECK INDEX.

See “Chapter 2-1. Invoking DB2 online utilities” on page 27 for an explanation of ways to execute DB2 utilities.

Data sets used by CHECK INDEX

Table 7 describes the data sets used by CHECK INDEX. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 7. Data sets used by CHECK INDEX

Data Set	Description	Required?
SYSIN	Input data set containing the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
Work data set	A temporary data set for collecting index key values to be checked. Its DD name is specified with the WORKDDN option of the utility control statement. The default DD name is SYSUT1. To find the approximate size in bytes of the work data sets, see page 74.	Yes
UTPRINT	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 card in the JCL:

Index space Object to be checked. It is named in the CHECK INDEX control statement and is accessed through the DB2 catalog. (If you want to check only one partition of an index, you must use the PART option in the control statement.)

Defining the work data set for CHECK INDEX: A single sequential data set, described by the DD statement specified in the WORKDDN option, is required during execution of CHECK INDEX.

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

1. For each table, multiply the number of records in the table by the number of indexes needing to be checked on the table.
2. Add the products obtained in step 1.
3. Add 9 to the length of the longest key.
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 sum the RBAs.

Creating the control statement

See “Syntax diagram” on page 72 for CHECK INDEX syntax and option descriptions. See “Sample control statements” on page 77 for examples of CHECK INDEX usage.

Instructions for specific tasks

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

Checking a single logical partition

You can run CHECK INDEX on a single logical partition of a nonpartitioning index. However, there are some limitations on what CHECK INDEX can detect:

- It 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
  ```

  and logical partition 2 might have these 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.

- It 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, then the keys within each partition are in sequence, but the keys for the index, as a whole, are out of sequence:

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

  When checking a single logical partition, this out of sequence condition is not detected.

Reviewing CHECK INDEX output

CHECK INDEX indicates whether or not a table space and its indexes are inconsistent, but 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 from CHECK INDEX.
2. Verify the point in time (TOLOGPOINT, TORBA, or TOCOPY) for each object 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, then run the RECOVER utility job again, including the table space and its indexes all in the same list.

Terminating or restarting CHECK INDEX

CHECK INDEX can be terminated in any phase without any integrity exposure.

You can restart a CHECK INDEX utility job, although it starts over again from the beginning.

For instructions on restarting a utility job, see “Restarting an online utility” on page 48.

Concurrency and compatibility

Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions of the same table space or index space are compatible.

Claims and drains: Table 8 shows which claim classes CHECK INDEX claims and drains and any restrictive state the utility sets on the target object.

Table 8. Claim classes of CHECK INDEX operations. Use of claims and drains; restrictive states set on the target object.

Target	CHECK INDEX	CHECK INDEX PART
Table space or partition	DW/UTRO	DW/UTRO
Partitioning index or index partition	DW/UTRO	DW/UTRO
Nonpartitioning index	DW/UTRO	
Logical partition of an index		DW/UTRO

Legend:

- DW: Drain the write claim class, concurrent access for SQL readers
- UTRO: Utility restrictive state, read only access allowed
- Blank: Object is not affected by this utility

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

Compatibility: Table 9 shows which utilities can run concurrently with CHECK INDEX on the same target object. 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 is also documented.

Table 9 (Page 1 of 2). CHECK INDEX compatibility

Action	CHECK INDEX
CHECK DATA	No
CHECK INDEX	Yes

Table 9 (Page 2 of 2). CHECK INDEX compatibility

Action	CHECK INDEX
CHECK LOB	Yes
COPY INDEXSPACE	Yes
COPY TABLESPACE	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 DUMP or VERIFY	Yes
REPAIR DELETE or REPLACE	No
REPORT	Yes
RUNSTATS	Yes
STOSPACE	Yes

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

Sample control statements

Example 1: Check all indexes in a sample table space.

```
//STEP1   EXEC DSNUPROC,UID='IUIQU1UQ.CHK1',
//         UTPROC='',
//         SYSTEM='V61A'
//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 DSN8D61A.DSN8S61E
//*
```

Example 2: Check one index. Check the project-number index (DSN8610.XPROJ1) on the sample project table.

```
CHECK INDEX (DSN8610.XPROJ1)
  SORTDEVT SYSDA
```

Example 3: Check more than one index. Check the indexes DSN8610.XEMPRAC1 and DSN8610.XEMPRAC2 on the employee to project activity sample table.

```
CHECK INDEX NAME (DSN8610.XEMPRAC1, DSN8610.XEMPRAC2)
```

Example 4: Check all indexes on a table space. Check all indexes on the employee-table table space (DSN8S61E).

```
CHECK INDEX (ALL) TABLESPACE DSN8S61E
  SORTDEVT 3380
```

Chapter 2-6. CHECK LOB

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

Run the CHECK LOB online utility against a LOB table space that is marked CHECK pending (CHKP) to identify structural defects. If none is found, the CHECK LOB utility turns the CHKP status off.

Run the CHECK LOB online utility against a LOB table space that is in auxiliary warning (AUXW) status to identify invalid LOBs. If none exists, the CHECK LOB utility turns AUXW status off.

Run CHECK LOB after a conditional restart or a point-in-time recovery on all table spaces where LOB table spaces might not be synchronized.

For a diagram of CHECK LOB syntax and a description of available options, see "Syntax and options of the control statement." For detailed guidance on running this utility, see "Instructions for running CHECK LOB" on page 81 .

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

Authorization required: To run this utility, the privilege set of this process must include 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:

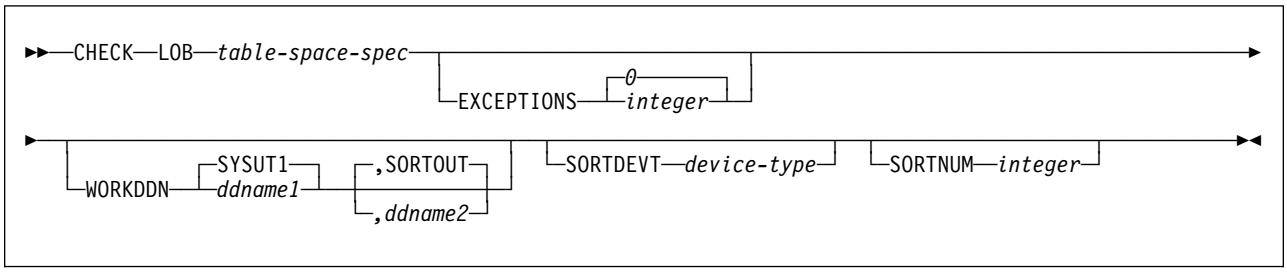
Phase	Description
UTILINIT	Initialization
CHECKLOB	Scans all active pages of the LOB table space
SORT	Sorts four types of records from the CHECKLOB phase; reports four times the number of rows sorted.
REPRTLOB	Examines records that are produced by the CHECKLOB phase and sorted by the SORT phase, and issues error messages
UTILTERM	Cleanup

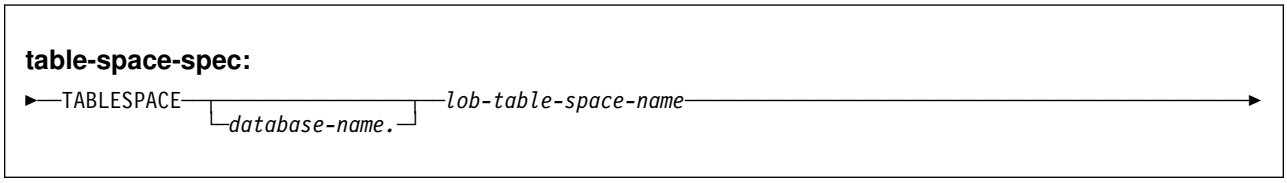
Syntax and options of the control statement

The utility control statement defines the function 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

For guidance in interpreting syntax diagrams, see "How to read the syntax diagrams" on page 4.





Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see "Control statement coding rules" on page 27.

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 **EXCEPTIONS 0,** which indicates **no limit** on the number of exceptions.

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.*

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. *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.*

If you omit SORTDEVT and a sort is required, you must provide the DD statements that the sort program requires for the temporary data sets.

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.

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.

Instructions for running CHECK LOB

To run CHECK LOB:

1. Read “Before running CHECK LOB” on page 82.
2. Prepare the necessary data sets, as described in “Data sets used by CHECK LOB” on page 82.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27. (For examples of JCL for CHECK LOB, see “Sample control statements” on page 84.)
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 83. (For a complete description of the syntax and options for CHECK LOB, see “Syntax diagram” on page 80.)
5. Check the compatibility table in “Concurrency and compatibility” on page 84 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 83.
7. Run CHECK LOB.

See “Chapter 2-1. Invoking DB2 online utilities” on page 27 for a description of ways to execute DB2 utilities.

Before running CHECK LOB

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

Data sets used by CHECK LOB

Table 10 describes the data sets that CHECK LOB uses. Include statements in your JCL for each required data set and any optional data sets you want to use.

Table 10. Data sets used by CHECK LOB

Data Set	Description	Required?
SYSIN	An input data set containing the utility control statement.	Yes
SYSPRINT	An output data set for messages.	Yes
Work data set	One temporary data set for sort input and sort output. The symbolic name of the DD statement is specified with the WORKDDN option of the utility control statement. The default *ddname* for this data set is SYSUT1. To find the approximate size in bytes of the work data sets, see page 82.	Yes
UTPRINT	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 to be checked. This object is named in the CHECK LOB control statement and is accessed through the DB2 catalog.

Defining work data sets: Two sequential data sets, described by the DD statements named in the WORKDDN option, are required during execution of CHECK LOB.

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

1. Find the *high allocated page number,* either from the NACTIVEF column of the SYSIBM.SYSTABLESPACE catalog table after running the RUNSTATS utility on the LOB table space, or from information in the VSAM catalog data set.
2. Use the formula *(43 × NACTIVEF × 4).* The resulting value is the approximate size, in bytes, of the work data set required.

Creating the control statement

See “Syntax diagram” on page 80 for CHECK LOB syntax and option descriptions. See “Sample control statements” on page 84 for examples of CHECK LOB usage.

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"
"Resetting CHECK pending status for a LOB table space"

Finding and resolving violations

CHECK LOB issues message DSNU743I whenever a LOB value 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 associated with the LOB (use the rowid given in message DSNU743I). For more information, see UPDATE or DELETE in Chapter 6 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® Support Center 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 found in the LOB table space using the REPAIR utility.
2. To reset CHECK pending or AUXW status, run CHECK LOB again, or run the REPAIR utility.

Terminating or restarting CHECK LOB

Terminating CHECK LOB

If you terminate CHECK LOB during the CHECKLOB phase, it sets the table space to CHECK pending status. Otherwise, CHECK LOB resets the CHECK pending status at the end of the phase if no errors are detected.

For instructions on terminating an online utility, see "Terminating an online utility with the TERM UTILITY command" on page 47.

Restarting CHECK LOB

You can restart a CHECK LOB utility job, but it starts from the beginning again.

For instructions on restarting a utility job, see "Restarting an online utility" on page 48.

Concurrency and compatibility

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

Table 11. Claim classes on a LOB table space and index on the auxiliary table for CHECK LOB operations. Use of claims and drains; restrictive states set on the target object.

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 control statements

Example: Checking a LOB table space. Check the table space TLIQUG02 in database DBIQUG01 for structural defects or invalid LOB values.

```
//STEP1    EXEC DSNUPROC,UID='IUIQU2UG.CHECKL',
//         UTPROC='',
//         SYSTEM='V61A'
//SYSERR   DD DSN=IUIQU2UG.STEP1.SYSERR,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSUT1   DD DSN=IUIQU2UG.CHECKL.SYSUT1,UNIT=SYSDA,
//         SPACE=(4000,(20,20),,,ROUND),
//         DISP=(MOD,DELETE,CATLG)
//SYSREC   DD DSN=CUST.FM.CSFT320.DATA,DISP=SHR,UNIT=SYSDA,
//         VOL=SER=123456
//SORTOUT  DD DSN=IUIQU2UG.STEP1.SORTOUT,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
CHECK LOB TABLESPACE DBIQUG01.TLIQUG02
      EXCEPTIONS 3 WORKDDN SYSUT1,SORTOUT SORTDEVT SYSDA
      SORTNUM 4
```

Chapter 2-7. 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

There are two types of image copies:

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

The copies are used by the RECOVER utility when recovering a table space or index space to the most recent time or to a previous time.

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. This provides a performance advantage, and allows the RECOVER utility to process the logs for all table spaces and index spaces in a single pass.

For a diagram of COPY syntax and a description of available options, see “Syntax and options of the control statement” on page 86. For detailed guidance on running this utility, see “Instructions for running COPY” on page 92.

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 available to the RECOVER utility. It is your installation's responsibility to ensure 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 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 the whole table space. DB2 resets the informational COPY pending status (ICOPY) after you copy an index space or index.

Related information: See Section 4 (Volume 1) of *DB2 Administration Guide* for uses of COPY in the context of planning for database recovery. For information about creating copies inline during LOAD, see “Using inline COPY with LOAD” on

page 160. You can also create inline copies during REORG; see “Using inline COPY with REORG TABLESPACE” on page 319 for more information.

Authorization required: To execute this utility, the privilege set of the process must include one of the following:

- 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 CONCURRENT option must provide the necessary authority to execute the DFDSS DUMP command.

Execution phases of COPY: The COPY utility operates in these phases:

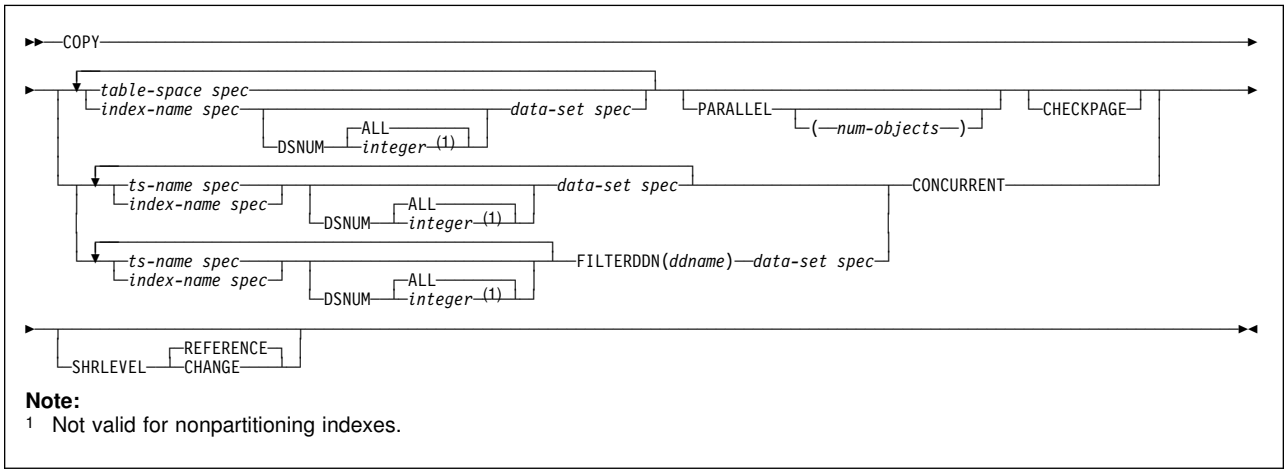
Phase	Description
UTILINIT	Initialization and setup
REPORT	Reporting for CHANGELIMIT option
COPY	Copying
UTILTERM	Cleanup

Syntax and options of the control statement

The utility control statement defines the function 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.

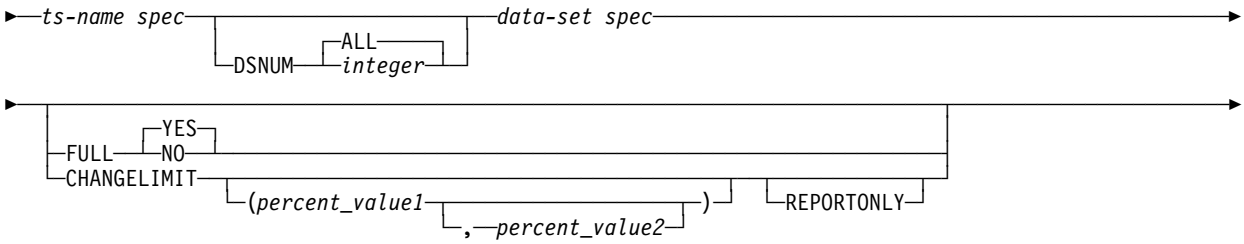
Syntax diagram

For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.

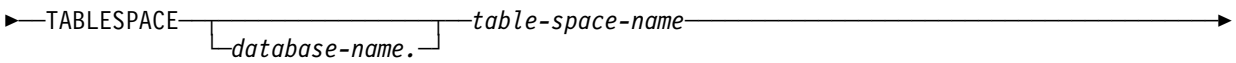


Note:
1 Not valid for nonpartitioning indexes.

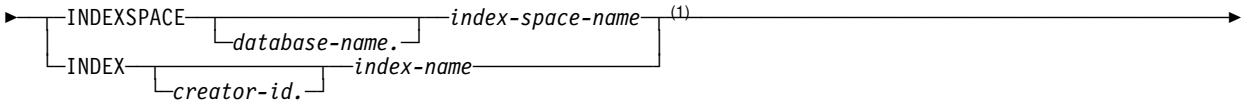
table-space spec:



ts-name spec:

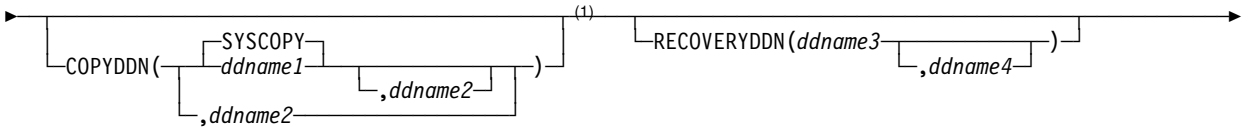


index-name spec:



Note:
1 INDEXSPACE is the preferred specification.

data-set spec:



Note:
1 If you specified a list of objects, only one object in the list can use the default (SYSCOPY); you must specify the DD names for the rest of the objects listed.

Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see “Control statement coding rules” on page 27.

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 the table space belongs to.

The **default** is **DSNDB04**.

table-space-name Is the name of the table space to be copied.

Specify a catalog or directory table space by itself in a single COPY statement, or only 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 index space specified must be defined with the COPY YES attribute.

database-name Optionally specifies the name of the database the index space belongs to.

The **default** is **DSNDB04**.

index-space-name Specifies the name of the index space to be copied.

INDEX *creator-id.index-name*
Specifies the index to be copied.

creator-id Optionally specifies the creator of the index.

The **default** is **DSNDB04**.

index-name Specifies the name of the index that is to be copied.

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, identifies a partition to be copied; or it copies the entire index space.

If a data set of a nonpartitioned table space is in the COPY pending status, you must copy the entire table space.

ALL Copies the entire table space or index space.

The **default** is **ALL**.

You must use the **ALL** default for a non-partitioning index.

integer Is the number of a partition or data set to be copied.

An integer value is not valid for non-partitioning indexes.

For a partitioned table space or index space, the integer is its partition number. The maximum is 254.

For a nonpartitioned table space or nonpartitioning index space, find the integer at the end of the data set name as cataloged in the VSAM catalog. The data set name has this format:

catname.DSNDB*x*.*dbname*.*spacename*.I0001.A*nnn*

where:

catname	The VSAM catalog name or alias
x	C or D
dbname	The database name
spacename	The table space or index space name
nnn	The data set integer.

If image copies are taken by data set (rather than by table space), then RECOVER or MERGECOPY must use the copies by data set. For a nonpartitioned table space, if image copies are taken by data set and you run MODIFY RECOVERY with DSNUM ALL, then the table space is placed in COPY pending status if a full image copy of the entire table space does not exist.

COPYDDN *ddname1,ddname2*

Specifies the DD statements for the primary (*ddname1*) and backup (*ddname2*) copied data sets for the image copy at the local site.

ddname is the DD name.

The **default** is **SYSCOPY** for the primary copy. You can only use the default for one object in the list.

If you use the CHANGELIMIT REPORTONLY option, you may use a DD DUMMY card when you specify the SYSCOPY output data set. This card prevents a data set from being allocated and opened.

It is recommended that you 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 already recorded in SYSIBM.SYSCOPY, a message is issued and no copy is made. If it identifies a cataloged data set with only the same name, no copy is made. For cataloged image copy data sets, CATLG must be specified for the normal termination disposition in the DD statement; for example, DISP=(MOD,CATLG,CATL). The DSVOLSER field of the SYSIBM.SYSCOPY entry will be blank.

When the image copy data set is going to a tape volume, the VOL=SER parameter should be specified on the DD statement.

If you use the CONCURRENT and FILTERDDN options, make sure the size of the copy data set is large enough to include all of the objects in the list.

RECOVERYDDN *ddname3,ddname4*

Specifies the DD statements for the primary (*ddname3*) and backup (*ddname4*) copied 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 as for COPYDDN.

If you use the CONCURRENT and FILTERDDN options, make sure the size of the copy data set is large enough to include all of the objects in the list.

PARALLEL Specifies the maximum number of objects in the list that should be processed in parallel. If you omit this keyword, the list is not processed in parallel.

(*num-objects***)** Specifies the number of objects in the list that should be processed in parallel. This value can be adjusted to a smaller value if COPY encounters storage constraints.

If you specify 0 or do not specify a value for *num-objects*, COPY determines the optimal number of objects to process in parallel.

CHECKPAGE If specified, checks each page in the table space or index space 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. If more than one error exists in a given page, only the first error is identified. COPY will continue checking the remaining pages in the table space or index space after an error is found.

FULL Makes either a full or an incremental image copy.

YES Makes 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 specified DSNUM.

The **default** is **YES**.

NO Makes only an incremental image copy. Only changes since the last image copy are 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 being copied.
- After a successful LOAD or REORG operation, unless an inline copy was made during the LOAD or REORG.
- The table space you specify is one of the following: DSNDB01.DBD01, DSNDB01.SYSUTILX, or DSNDB06.SYSCOPY.

COPY automatically takes a full image copy of a table space if you specify FULL NO when an incremental image copy is not allowed.

FILTERDDN *ddname*

Specifies the DD statement for the filter data set to be used, if desired, by COPY with the CONCURRENT option. COPY uses this data set to automatically build a list of table spaces to be copied by DFSMSdds™ with one DFSMSdds "DUMP" statement.

If FILTERDDN is specified, the SYSIBM.SYSCOPY records for all objects in the list will 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. Image copies taken using SHRLEVEL CHANGE are not recommended for use with RECOVER TOCOPY.

SHRLEVEL CHANGE is not allowed when you use DFSMS Concurrent Copy for table spaces having a page size greater than 4KB.

CONCURRENT Executes DFSMS concurrent copy to make the full image copy. The image copy is recorded in SYSCOPY with ICTYPE=F and STYPE=C.

If the SYSPRINT DD card points to a data set, you must use a DSSPRINT DD card.

When SHRLEVEL(REFERENCE) is specified, an ICTYPE=Q record is placed into the SYSCOPY table after the object has been quiesced. If COPY fails, then this record remains in SYSCOPY. When COPY is successful, then this ICTYPE=Q record is replaced with the ICTYPE=F record.

For table spaces with a 32 KB page size, you must run the job with the SHRLEVEL REFERENCE (default) option when using the CONCURRENT option. Otherwise, the job is terminated, and message DSNU423I is issued.

CHANGELIMIT Specifies the percent limit of changed pages in the table space, partition, or data set when an incremental or full image copy should be taken.

percent_value1 Specifies a value in the CHANGELIMIT range. *percent_value1* must be an integer or decimal value from 0 to 100. You do not need to specify leading zeroes, and the decimal point is not required when specifying a whole integer. Specify one decimal place for a decimal value (for example, .5).

percent_value2 Specifies the second value in the CHANGELIMIT range. *percent_value2* must be an integer or decimal value from 0 to 100. You do not need to specify leading zeroes, and the decimal point is not required when specifying a whole integer.

COPY CHANGELIMIT accepts values in any order.

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 change 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 value specified and less than the highest value specified.
- Creates a full image copy if the percentage of changed pages is equal to or greater than the highest value specified.
- Does not create an image copy if the percentage of changed pages is less than or equal to the lowest value specified.
- If both values are equal, creates a full image copy if the percentage of changed pages is equal to or greater than the value specified.

The default values are *(1,10)*.

You cannot use the CHANGELIMIT option for a table space or partition 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 displayed. If you specify the REPORTONLY option, then only image copy information is displayed. Image copies are not taken, only recommended.

Instructions for running COPY

To run COPY, you must:

1. Read “Before running COPY” on page 93 in this chapter.
2. Prepare the necessary data sets, as described in “Data sets used by COPY” on page 93.

3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27. (For examples of JCL for COPY, see “Sample control statements” on page 108.)
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 95. (For a complete description of the syntax and options for COPY, see “Syntax and options of the control statement” on page 86.)
5. Check the compatibility table in “Concurrency and compatibility” on page 106 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 105.
7. Run COPY.

See “Chapter 2-1. Invoking DB2 online utilities” on page 27 for an explanation of ways to execute DB2 utilities.

Before running COPY

Checking table space status: You cannot copy a table space that is in the CHECK pending or RECOVER pending status. See “Resetting RECOVER pending or REBUILD pending status” on page 248 for information about resetting these statuses.

Resetting COPY pending 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 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.

Data sets used by COPY

Table 12 describes the data sets required for COPY. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 12 (Page 1 of 2). Data sets used by COPY

Data Set	Description	Required?
SYSIN	Input data set containing the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
DSSPRINT	Output data set for messages; required when CONCURRENT copy is used and the SYSPRINT DD card points to a data set.	No
Filter	A single data set DB2 uses when you specify the FILTERDDN option in the utility control statement; contains a list of VSAM data set names built by DB2, and is used during COPY when the CONCURRENT and FILTERDDN options are specified.	Yes[1]

Table 12 (Page 2 of 2). Data sets used by COPY

Data Set	Description	Required?
Copies	From one to four output data sets to contain the resulting image copy data sets. Their DD names are specified with the COPYDDN and RECOVERYDDN options of the utility control statement. The default is one copy, in the data set described by the SYSCOPY DD statement.	Yes

Note: 1 Required if you specify the FILTERDDN option.

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

Table space or Index space
Object to be copied. It is named in the COPY control statement and is accessed through the DB2 catalog. (If you want to copy only certain data sets in a table space, you must use the DSNUM option in the control statement.)

Catalog COPY records each copy in the DB2 catalog table SYSIBM.SYSCOPY.

Output data set size: Image copies are written to sequential non-VSAM data sets. To find the approximate size of the image copy data set for a table space, in bytes, you can execute COPY with the CHANGELIMIT REPORTONLY option, or use the following procedure:

1. Find the *high allocated page number,* either from the NACTIVE 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: Use the formula *(240 + (80 × n))* to determine the approximate FILTER data set size required, in bytes, where *n* = the number of objects specified in the COPY control statement.

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 buffering with the BUFNO parameter; for example, you might specify BUFNO=30. See also “Data sets used by online utilities” on page 28 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 named by the COPYDDN option. After the image copy is taken, the DSVOLSER column of the row inserted into SYSIBM.SYSCOPY contains blanks.

Duplicate image copy data sets are not allowed. If there is a cataloged data set already recorded in 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 SYSCOPY, it uses the MVS 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, it must use correspondingly more of the log to recover. It is to your benefit, and it is your responsibility, to keep the MVS catalog consistent with SYSIBM.SYSCOPY about existing image copy data sets.

Creating the control statement

See “Syntax diagram” on page 86 for COPY syntax and option descriptions. See “Sample control statements” on page 108 for examples of COPY usage.

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

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 makes a full image copy of the DSN8S61E table space in database DSN8D61A:

```
COPY TABLESPACE DSN8D61A.DSN8S61E
```

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 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. We recommend taking a full image copy after CREATE or LOAD operations for a new object that is populated, after a REORG operation for an existing object, and after LOAD RESUME of an existing object.

We recommend copying 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 to be applied during recovery. At 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, “Resetting an advisory or restrictive status” on page 527.

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 was NO, the COPY pending status is set for the table space. A full image copy must be made for any subsequent recovery of the data. An incremental image copy is not allowed.

If the LOG option was 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 will automatically take 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.SYSUTILX 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, it is most efficient to defer copying the catalog table or directories until the other copy jobs have completed. 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:

- 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) if a full image copy of the table space exists, or if 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. Moreover, the full image copy must have been made after the most recent application to the table space of CREATE, REORG or LOAD, or it must be an inline copy made during the most recent application of LOAD or REORG.

Sample control statement: To specify an incremental image copy, use FULL NO on the COPY statement, as in this example:

```
COPY TABLESPACE DSN8D61A.DSN8S61E
  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 was 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, 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 was defined with the TRACKMOD YES option. Incremental image copies of a table space that was defined with TRACKMOD NO will still save space, at some performance cost.

Restrictions: You cannot make incremental copies of the DSNDB01.DBD01 and copies of table space DSNDB01.SYSUTILX in the directory, 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

You can use a single invocation of the COPY utility to 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 installed with the option RECOVERYSITE). All copies are identical, and are produced at the same time from one invocation of COPY.

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 the backup copied data set. The ICUNIT column in SYSIBM.SYSCOPY specifies whether the image copy data set is on tape or DASD.

Remote site recovery: In preparation for remote site recovery, system and application libraries and the DB2 catalog and directory are assumed to be 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 data for remote site recovery current. This information can be kept on tape until needed.

Naming the data sets for the copies: The option COPYDDN of COPY names the output data sets that receive copies for local use. The option RECOVERYDDN of COPY names the output data sets that receive copies intended for remote site recovery. The options have these formats:

```
COPYDDN (ddname1,ddname2)

RECOVERYDDN (ddname3,ddname4)
```

The ddnames 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 DSN8S61E in database DSN8D61A, using LOCALDD1 and LOCALDD2 as ddnames for the primary and backup copies used on the local system and RECOVDD1 and RECOVDD2 as ddnames for the primary and backup copies for remote site recovery:

```
COPY TABLESPACE DSN8D61A.DSN8S61E
  COPYDDN (LOCALDD1,LOCALDD2)
  RECOVERYDDN (RECOVDD1,RECOVDD2)
```

You do not have to make copies for local use and for remote site recovery at the same time. COPY allows you to use either option COPYDDN or option RECOVERYDDN without the other. If you make copies for local use more often than copies for remote site recovery, then a remote site recovery might work from an older copy, and more of the log, than a local recovery; hence, it would take longer. But, in your plans for remote site recovery, that difference might be acceptable. You can also use MERGECOPY RECOVERYDDN to create recovery site full copies, and merge local incrementals into new recovery site full copies.

Making multiple incremental image copies: DB2 cannot make incremental image copies if:

- 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 SYSCOPY table, and issues this 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 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 insure 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 will still be 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.

However, creating copies simultaneously will not provide you with a consistent recovery point unless you follow up with a QUIESCE of the table space.

Copying a list of objects

Within a single COPY statement, the COPY utility allows you to process a list containing 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 after 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 will have a REPORT phase, so the phase will switch 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 will be reset.
- Using the SHRLEVEL(REFERENCE) option:

  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 SYSIBM.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 will have identical RBA or LRSN values for the START_RBA column for the SYSIBM.SYSCOPY rows: the current LRSN at the end of the COPY phase.
- Using the SHRLEVEL(CHANGE) option:

  Claims the read class for each table space and index space. The claim initializes before the copy of the object starts, and releases when the copy on the object completes.

Utility processing inserts a SYSIBM.SYSCOPY row for objects in the list when the copy of each object is complete.

Objects in the list will have different LRSN values for the START_RBA column for the SYSIBM.SYSCOPY rows: 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 DASD devices. If COPY encounters a tape volume in the list, processing of remaining objects pauses until the tape object has completed, then parallel processing resumes.

You can have DFSMS copy a list of table spaces. Certain table spaces cannot be included in a list of table spaces with the SHRLEVEL REFERENCE option; each one of the following table spaces must be specified as a single object:

DSNDB01.SYSUTILX
DSNDB06.SYSCOPY
DSNDB01.SYSLGRNX

The only exception 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 has executed 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.

If a job step containing more than one COPY statement abends, **do not use TERM UTILITY.** Restart the job from the last commit point using RESTART instead. Terminating COPY 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 DFSMS concurrent copy

You might be able to gain improved availability by using the Concurrent Copy function of Data Facility Storage Management Subsystem (DFSMS). You can subsequently run the RECOVER utility to restore those image copies and apply the necessary log records to them to complete recovery.

The CONCURRENT option invokes DFSMS concurrent copy. The COPY utility records the resulting DFSMS concurrent copies in the catalog table SYSIBM.SYSCOPY with ICTYPE=F and STYPE=C.

To obtain a consistent offline backup:

1. Start the DB2 objects being backed up for read-only access by issuing the following command:

   ```
   -START DATABASE(database name) SPACENAM(tablespace-name) ACCESS(RO)
   ```

This 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 being backed up.
3. Back up the DB2 data sets if 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 ddname, a RECOVERYDDN ddname, or both.
- If the SYSPRINT DD card points to a data set, you must use a DSSPRINT DD card.
- You must use the SHRLEVEL REFERENCE option for table spaces with a 8KB, 16KB, or 32KB page size.

Restrictions on using DFSMS concurrent copy: You cannot use a copy made with DFSMS concurrent copy with the PAGE or ERRORRANGE options. If you specify PAGE or ERRORRANGE, RECOVER bypasses any concurrent copy records when searching the SYSCOPY table for a recoverable point.

You cannot use the CONCURRENT option with SHRLEVEL CHANGE on a table space with 8 KB, 16 KB, or 32 KB page size.

Also, you cannot run the following DB2 stand-alone utilities on copies made by DFSMS 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 DFSMS concurrent copy: To use COPY to take DFSMS concurrent copies, you must have the following hardware and software:

- OS/390 Release 3
- 3990 model 3 or 3990 model 6 controller at the extended platform attached to the DASD. A COPY job fails if one or more of the table spaces names is on DASD that does not have the controller.

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 output device, specify FILTERDDN in your COPY statement to improve table space availability. In this scenario, specifying COPY without the FILTERDDN option forces DFSMS to process the list of table spaces sequentially, which might limit the availability of some of the table spaces 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 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 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 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 copied if an incremental image copy is taken.
- The percentage of changed pages.
- The type of image copy 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 percent of changed pages is greater than the low CHANGELIMIT and less than the high CHANGELIMIT value.
3 (informational) If the percent of changed pages is greater than or equal to the high CHANGELIMIT value.

If you specify multiple copy statements in one job step, that job step will report 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 against the entire list of COPY statements 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 card to DD DUMMY so no output data set is allocated.

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 first step.

Preparing for 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, read the following topics pertaining to recovery.

Using incremental copies: The RECOVER TABLESPACE utility merges all incremental image copies since the last full image copy, and must have all the image copies available at the same time. If there is any likelihood that the requirement will 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 there are not enough tape units, RECOVER TABLESPACE can still attempt to recover the object. RECOVER dynamically allocates the full image copy and attempts to allocate dynamically 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 an incremental copy cannot be allocated, 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: Primary and secondary image copies are recommended after a LOAD or REORG operation specified with LOG NO when an inline copy is not created, so 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 which contains all referentially related structures, you do not need
| to QUIESCE these objects first in order to create a consistent point of recovery.

You should quiesce and copy both the base table space and the LOB table space at the same time to establish a recoverable point of consistency. Be aware that QUIESCE does not create a recoverable point for a LOB table space that contain LOBs defined with LOG NO.

| ***Setting and clearing the informational COPY pending status:*** The following
| utilities can place an index that was defined with the COPY YES attribute 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 index space is recoverable using the RECOVER utility. If you need to recover an index that did not have a full image copy taken, use the REBUILD INDEX utility to rebuild data from the table space.

Improving performance

A full image copy and subsequent incremental image copies can be merged into a new full copy by running MERGECOPY. After reorganizing a table space, the first image copy *must* be a full image copy.

The decision whether to run a full or an incremental image copy must not be based on the number of rows updated since the last image copy was taken. Instead, it must be based on the percentage of pages containing at least one updated record (not the number of records updated). 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 or incremental copy is required; see "Specifying conditional image copies" on page 102 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.

Considerations for running COPY

This section describes additional points to keep in mind when running COPY.

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 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

We recommend using 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 cover 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 to keep. Use NOEMPTY to avoid deleting all the data sets from the integrated catalog facility catalog when the limit is reached.

The high-level qualifier of the data set name might not be the catalog name or its alias. In that case, include a DD statement for JOBCAT or STEPCAT in the COPY job, and in the eventual RECOVER job, to tell what catalog the data set is in.

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 results in the following:

- 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 102 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.

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 un-cataloged 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

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 for a LOB table space.

Terminating or restarting COPY

For instructions on restarting a utility job, see “Restarting an online utility” on page 48.

Warning against TERM UTILITY

We do *not* recommend stopping a COPY job with the TERM UTILITY command. 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 COPY had started processing, but not yet completed. For copies made with SHRLEVEL REFERENCE, it is possible that some objects in the list might not have a 'T' record. For SHRLEVEL CHANGE, some objects might have a valid 'F', 'I', or 'T' record, or no record at all. The COPY utility does not allow you to take an incremental image copy if a 'T' record exists. To reset the status, you must make a full image copy.

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 statements
- Is valid for a list of objects
- Requires a minimum of re-processing
- Keeps the DB2 catalog and the integrated catalog facility catalog in agreement

DB2 uses the same image copy data set when you RESTART from the last commit point. Therefore, specify DISP=(MOD,CATLG,CATLG) on your DD statements. You cannot use RESTART(PHASE) for any COPY job.

Implications of DISP on the DD statement

If you terminate a COPY job that uses the parameter DISP=(MOD,CATLG,CATLG), then:

- If there is only one COPY statement, no row is written to SYSIBM.SYSCOPY, but an image copy data set has been created and is cataloged in the integrated catalog facility catalog. You should delete that data set.
- If there are several COPY statements in one COPY job step, a row for each successfully completed copy is written into SYSIBM.SYSCOPY. However, all the image copy data sets have been created and cataloged. You should delete all image copy data sets not recorded in SYSIBM.SYSCOPY.

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 a COPY job

You cannot use RESTART(PHASE) for any COPY job. If you do *not* use the -TERM UTILITY command, you can use RESTART to restart the job from the last commit point of the current table space or index space.

Restarting COPY after an out of space condition

See “Restarting after the output data set is full” on page 49 for guidance in restarting COPY from the last commit point after receiving an out of space condition.

Concurrency and compatibility

Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions of the same table space or index space are compatible.

Claims and drains

Table 13 on page 107 shows which claim classes COPY claims and drains and any restrictive status the utility sets on the target table space.

Table 13. Claim classes of COPY operations. Use of claims and drains; restrictive states set on the target object.

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 *searched* DELETE without the WHERE clause.

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

Compatibility

Table 14 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 is also shown.

Table 14 (Page 1 of 2). COPY compatibility

Action	COPY INDEXSPACE SHRLEVEL REFERENCE	COPY INDEXSPACE SHRLEVEL CHANGE	COPY TABLESPACE SHRLEVEL REFERENCE	COPY TABLESPACE SHRLEVEL CHANGE
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
DIAGNOSE	Yes	Yes	Yes	Yes
LOAD	No	No	No	No
MERGECOPY	No	No	No	No
MODIFY	No	No	No	No
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

Table 14 (Page 2 of 2). COPY compatibility

Action	COPY INDEXSPACE SHRLEVEL REFERENCE	COPY INDEXSPACE SHRLEVEL CHANGE	COPY TABLESPACE SHRLEVEL REFERENCE	COPY TABLESPACE SHRLEVEL CHANGE
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 TABLESPACE PAGE REPLACE	Yes	Yes	No	No
REPAIR LOCATE INDEX PAGE REPLACE	No	No	Yes	No
REPORT	Yes	Yes	Yes	Yes
RUNSTATS INDEX	Yes	Yes	Yes	Yes
RUNSTATS TABLESPACE	Yes	Yes	Yes	Yes
STOSPACE	Yes	Yes	Yes	Yes

To run on DSNDB01.SYSUTILX, COPY must be the only utility in the job step. Further, 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 control statements

In some cases, a COPY utility job might be run more than once. To facilitate avoiding 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 87 for further information.

Example 1: Full image copy. Make a full image copy of table space DSN8S61E in database DSN8D61A.

```
//STEP1    EXEC DSNUPROC,UID='IUJMU111.COPYTS',
//         UTPROC='',
//         SYSTEM='V61A',DB2LEV=DB2A
//SYSIN    DD *
//SYSCOPY  DD DSN=COPY001F.IFDY01,UNIT=SYSDA,VOL=SER=CPY01I,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//SYSIN DD *
COPY TABLESPACE DSN8D61A.DSN8S61E
```

Example 2: Full image copies of a list of objects. Make full image copies at the local site and recovery site, along with backups, of the following objects:

- table space DSN8D61A.DSN8S61D, and its indexes:
  - DSN8610.XDEPT1
  - DSN8610.XDEPT2
  - DSN8610.XDEPT3
- table space DSN8D61A.DSN8S61E, and its indexes

Do not allow updates during the copy, and process four objects in parallel. As the copy of an object completes, the next object in the list begins processing in parallel until all the objects have been processed.

This COPY job creates a point of consistency for the table spaces and their indexes. You can subsequently use the RECOVER utility with the TOCOPY option to recover all of these objects; see page 253 for an example.

```
COPY
   TABLESPACE DSN8D61A.DSN8S61D
        COPYDDN (COPY1,COPY2)
        RECOVERYDDN (COPY3,COPY4)
  INDEX DSN8610.XDEPT1
        COPYDDN (COPY5,COPY6)
        RECOVERYDDN (COPY7,COPY8)
  INDEX DSN8610.XDEPT2
        COPYDDN (COPY9,COPY10)
        RECOVERYDDN (COPY11,COPY12)
  INDEX DSN8610.XDEPT3
        COPYDDN (COPY13,COPY14)
        RECOVERYDDN (COPY15,COPY16)
  TABLESPACE DSN8D61A.DSN8S61E
        COPYDDN (COPY17,COPY18)
        RECOVERYDDN (COPY19,COPY20)
  INDEX DSN8610.XEMP1
        COPYDDN (COPY21,COPY22)
        RECOVERYDDN (COPY23,COPY24)
  INDEX DSN8610.XEMP2
        COPYDDN (COPY25,COPY26)
        RECOVERYDDN (COPY27,COPY28)
 PARALLEL(4)
 SHRLEVEL REFERENCE
```

Example 3: Copies for local site and recovery site. Make full image copies of table space DSN8S61C in database DSN8D61P at the local site and the recovery site.

```
//STEP1    EXEC DSNUPROC,UID='IUJMU111.COPYLST',
//         UTPROC='',
//         SYSTEM='V61A',DB2LEV=DB2A
//COPY1    DD DSN=IUJMU111.COPYLST.STEP1.COPY1,
//         DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//         SPACE=(4000,(20,20),,,ROUND)
//COPY2    DD DSN=IUJMU111.COPYLST.STEP1.COPY2,
//         DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//         SPACE=(2000,(20,20),,,ROUND)
//COPY3    DD DSN=IUJMU111.COPYLST.STEP1.COPY3,
//         DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//         SPACE=(2000,(20,20),,,ROUND)
//COPY4    DD DSN=IUJMU111.COPYLST.STEP1.COPY4,
//         DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//         SPACE=(2000,(20,20),,,ROUND)
//SYSIN    DD *
 COPY TABLESPACE DSN8D61P.DSN8S61C
       COPYDDN     (COPY1,COPY2)
       RECOVERYDDN (COPY3,COPY4)
```

Example 4: Incremental copy with updates allowed. Make incremental image copies of table space DSN8S61D in database DSN8D61A, allowing update activity to occur during the copy process.

```
//STEP1    EXEC DSNUPROC,UID='IUJMU111.COPYLSTI',
//         UTPROC='',
//         SYSTEM='V61A',DB2LEV=DB2A
//SYSCOPY  DD DSN=IUJMU111.COPYLSTI.STEP1.CPY01I,
//         DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//         SPACE=(4000,(20,20),,,ROUND)
//COPY2    DD DSN=IUJMU111.COPYLSTI.STEP1.CPY02I,
//         DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//         SPACE=(2000,(20,20),,,ROUND)
//COPY3    DD DSN=IUJMU111.COPYLSTI.STEP1.CPY03I,
//         DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//         SPACE=(2000,(20,20),,,ROUND)
//COPY4    DD DSN=IUJMU111.COPYLSTI.STEP1.CPY04I,
//         DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//         SPACE=(2000,(20,20),,,ROUND)
//SYSIN    DD *
COPY TABLESPACE DSN8D61A.DSN8S61D
     COPYDDN      (SYSCOPY,COPY2)
     RECOVERYDDN  (COPY3,COPY4)
     FULL NO
     SHRLEVEL CHANGE
```

Example 5: Invoking DFSMS concurrent copy with the COPY utility. Copy a table space, using the CONCURRENT option to execute DFSMS™ concurrent copy. Use a DSSPRINT DD card for message output.

```
//COPY     EXEC DSNUPROC,SYSTEM=V61A
//SYSCOPY1 DD DSN=COPY1,DISP=(NEW,CATLG,CATLG),
//         SPACE=(4000,(20,20),,,ROUND),UNIT=SYSDA,VOL=SER=DB2CC5
//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 *
COPY TABLESPACE DBASE1AA.TABLESPC
  COPYDDN     (SYSCOPY1)
  CONCURRENT
```

Example 6: Invoking DFSMS concurrent copy with the COPY utility using FILTER. Copy a list of table spaces, using the CONCURRENT and FILTERDDN options to create a single "DUMP" statement for DFSMS concurrent copy, allowing maximum availability.

```
//SYSCOPY  DD DSN=CONCOPY.WFILT,DISP=(MOD,CATLG,DELETE),
//            UNIT=SYSDA,SPACE=(CYL,(42,5),RLSE)
//FILT     DD DSN=FILT.TEST1,DISP=(MOD,CATLG,DELETE),
//            UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SYSIN    DD *
COPY TABLESPACE TS1
     TABLESPACE TS2
     TABLESPACE TS3
FILTERDDN(FILT)
COPYDDN(SYSCOPY)
CONCURRENT
    SHRLEVEL REFERENCE
```

Example 7: Invoking DFSMS concurrent copy with a list. Copy a list of table spaces, using the CONCURRENT option to execute DFSMS concurrent copy. Allow update activity during the COPY operation.

```
//STEP1    EXEC DSNUPROC,UID='IUJMU111.COPYLST',
//         UTPROC='',
//         SYSTEM='V61A',DB2LEV=DB2A
//COPY1    DD DSN=IUJMU111.COPYLST.STEP1.TS1,
//         DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//         SPACE=(4000,(20,20),,,ROUND)
//COPY2    DD DSN=IUJMU111.COPYLST.STEP1.TS2,
//         DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//         SPACE=(2000,(20,20),,,ROUND)
//COPY3    DD DSN=IUJMU111.COPYLST.STEP1.TS3,
//         DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//         SPACE=(2000,(20,20),,,ROUND)
//SYSIN    DD *
COPY TABLESPACE DBAU2901.TPAU2901
     COPYDDN(COPY1)
     TABLESPACE DBAU2901.TLAU2902
     COPYDDN(COPY2)
     TABLESPACE DBAU2901.TSAU2903
     COPYDDN(COPY3)
     CONCURRENT SHRLEVEL CHANGE
```

Example 8: Report image copy information for a table space. Recommend a full image copy if the percent of changed pages is equal to or greater than 40 percent. Recommend an incremental image copy if the percent of changed pages

is greater than 10 and less than 40 percent. Recommend no image copy if the percent of changed pages is 10 percent or less.

```
COPY TABLESPACE DSN8D61P.DSN8S61C CHANGELIMIT(10,40) REPORTONLY
```

Example 9: Make a conditional image copy. Take a full image copy of a table space if the number of changed pages is equal to or greater than 5 percent. Take an incremental image copy if the percent of changed pages is greater than 0 and less than 5 percent. If no pages have changed, do not take an image copy.

```
COPY TABLESPACE DSN8D61P.DSN8S61C CHANGELIMIT(5)
```

Example 10: Copying LOB table spaces together with related objects. Take a full image copy of base table space TPIQUD01 and LOB table spaces TLIQUDA1, TLIQUDA2, TLIQUDA3, and TLIQUDA4 in database DBIQUD01 if the number of changed pages is equal to or greater than the decimal percentage values specified for each object. Take an incremental image copy if the percent of changed pages falls in the range between the specified decimal percentage values. If no pages have changed, do not take an image copy. Also 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
```

Chapter 2-8. DIAGNOSE

The DIAGNOSE online utility generates information useful in diagnosing problems. It is intended to be used only under the direction of your IBM Support Center.

Interpreting 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 a diagram of DIAGNOSE syntax and a description of available options, see "Syntax and options of the control statement." For detailed guidance on running this utility, see "Instructions for running DIAGNOSE" on page 117 .

Authorization required: To execute this utility, the privilege set of the process must include one of the following:

- 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.

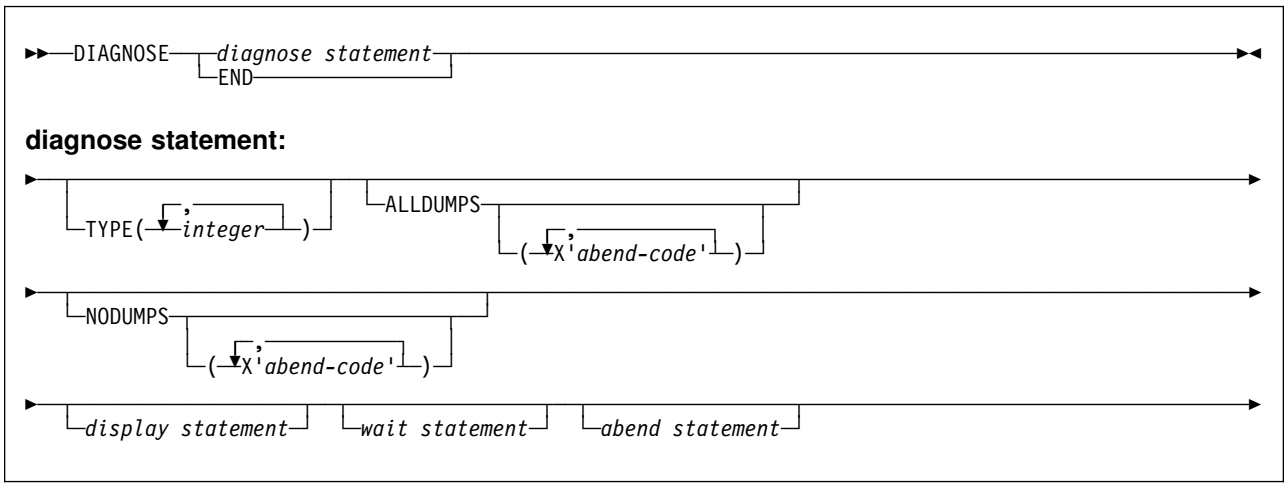
An ID with installation SYSADM authority can execute the DIAGNOSE utility with the WAIT statement option.

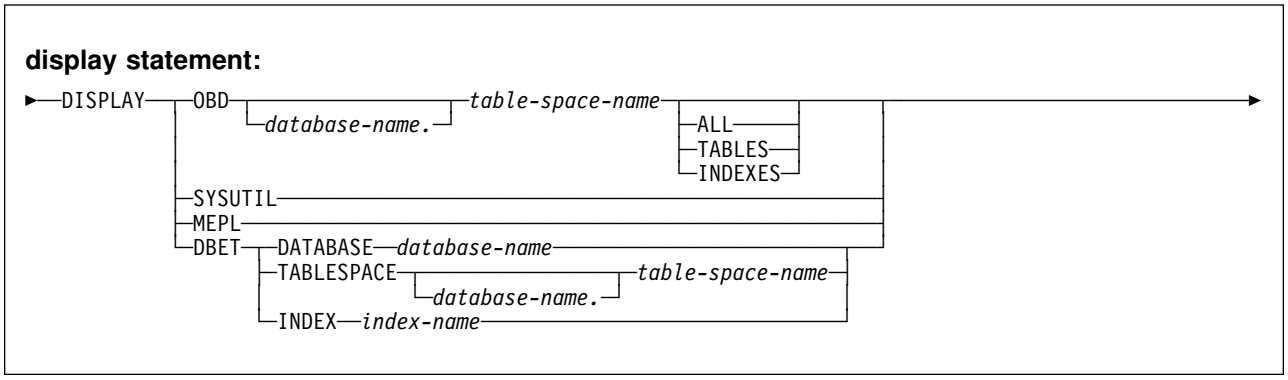
Syntax and options of the control statement

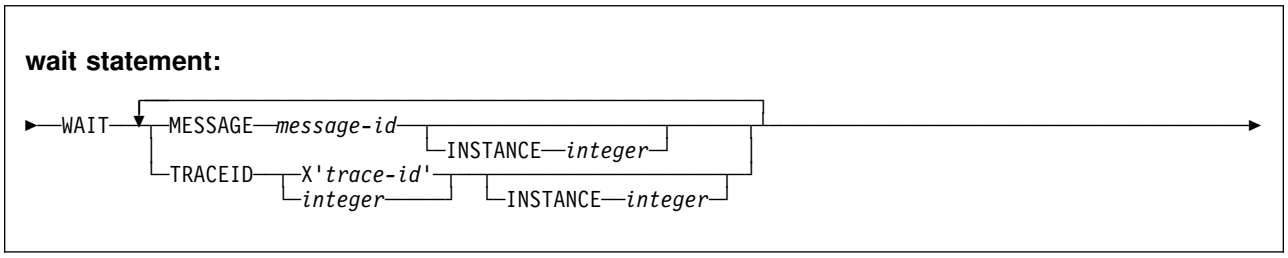
The utility control statement defines the function the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name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

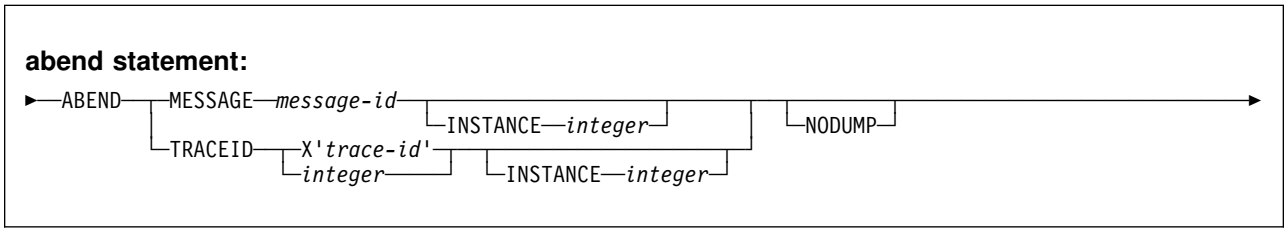
Syntax diagram

For guidance in interpreting syntax diagrams, see "How to read the syntax diagrams" on page 4.









Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see “Control statement coding rules” on page 27.

END Ends DIAGNOSE processing.

TYPE(*integer***, ...)**
Specifies one or more types of diagnose you wish to perform.

integer is the number of types of diagnoses. The maximum number of types is 32. See *DB2 Diagnosis Guide and Reference* for a list of available types.

ALLDUMPS(X'*abend-code***', ...)**
Forces a dump to be taken from 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 relationship 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 SYSUTIL record.

MEPL Dumps the module entry point lists (MEPLs) to SYSPRINT.

DBET Dumps the contents of a database exception table (DBET) to SYSPRINT. This option is intended to be used only under the direction of your IBM Support Center.

DATABASE *database-name*
Dumps the DBET entry associated with the specified database.

database-name is the name of the database.

TABLESPACE *database-name.table-space-name*
Dumps the DBET entry 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 associated with the specified index.

creator-name is the ID of the creator of the index.

index-name is the name of the index.

WAIT After encountering the specified utility message or utility trace ID, a message is issued to the console and utility execution is suspended until the operator replies to that message, the utility job times out, or the utility job is canceled. This allows events to be synchronized while 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, WAIT processing continues.

ABEND After encountering the specified utility message or utility trace ID, an abend is forced during utility execution.

If neither the utility message nor the trace ID are encountered, ABEND processing continues.

NODUMP
Suppresses the dump 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. Valid message IDs can be found in Section 3 of *DB2 Messages and Codes*.

message-id is the message in the form of U*xxx* or U*xxxx*.

INSTANCE *integer*
Specifies that a wait or an abend occurs 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 the message is encountered.

integer is the number of times 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. Valid trace IDs can be found in data set *prefix*.SDSNSAMP(DSNWEIDS).

trace-id is a trace ID associated with the utility trace (RMID21), and can be specified in either decimal (*integer*) or hexadecimal (X'*trace-id*').

INSTANCE *integer*
Specifies that a wait or an abend occurs when the TRACEID option has been encountered a specified number of times. If INSTANCE is not specified, a wait or abend occurs each time the trace ID is encountered.

integer is the number of times a trace ID is to be encountered before a wait or an abend occurs.

Instructions for running DIAGNOSE

To run DIAGNOSE, you must:

1. Prepare the necessary data sets, as described in "Data sets used by DIAGNOSE."
2. Create JCL statements, by using one of the methods described in "Chapter 2-1. Invoking DB2 online utilities" on page 27. (For examples of JCL for DIAGNOSE, see "Sample control statements" on page 118.)
3. Prepare a utility control statement, specifying the options for the tasks you want to perform, as described in "Instructions for specific tasks." (For a complete description of the syntax and options for DIAGNOSE, see "Syntax and options of the control statement" on page 113 .)
4. Run DIAGNOSE.

See "Chapter 2-1. Invoking DB2 online utilities" on page 27 for an explanation of ways to execute DB2 utilities.

Data sets used by DIAGNOSE

Table 15 describes the data sets used by DIAGNOSE. Include statements in your JCL for each required data set.

Table 15. Data sets used by DIAGNOSE

Data Set	Description	Required?
SYSIN	Input data set containing 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 cards in the JCL:

Database Database to gather diagnosis information about.

Table space
Table space to gather diagnosis information about.

Index space
Index to gather diagnosis information about.

Instructions for specific tasks

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

Forcing a utility abend with DIAGNOSE

DIAGNOSE can force an 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 by using the MESSAGE option of the ABEND statement.

Instead of using a message, you can use the TRACEID option of the ABEND statement to specify a trace IFCID associated with the utility to force an abend.

Use the INSTANCE keyword to specify the number of times the specified message or trace record is generated before the utility abends.

Terminating or restarting DIAGNOSE

You can terminate DIAGNOSE with the TERM UTILITY command.

Concurrency and compatibility

DIAGNOSE can run concurrently on the same target object with any SQL operation or utility, except a utility running on DSNDB01.SYSUTILX.

Sample control statements

Example 1: Sample JCL for DIAGNOSE.

```
//STEP1    EXEC DSNUPROC,UID='IUJMU116.COPY1',
//         UTPROC='',
//         SYSTEM='V61A'
//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 DSN8D61A.DSN8S61E
                     COPYDDN SYSCOPY1
DIAGNOSE
           END
//*
```

Example 2: Force dump for utility abend. Force a dump for any utility abend that occurs during the execution of the COPY utility.

```
DIAGNOSE
 ALLDUMPS
 COPY TABLESPACE DSNDB06.SYSDBASE
DIAGNOSE END
```

Example 3: Force utility abend if message is issued. Abend the LOAD utility the fifth time message DSNU311 is issued. Do not generate a dump.

```
DIAGNOSE
 ABEND MESSAGE U311 INSTANCE 5 NODUMP
LOAD DATA RESUME NO
  INTO TABLE TABLE1
   (NAME POSITION(1) CHAR(20))
DIAGNOSE END
```

Example 4: Display SYSUTIL table. Display all rows in the SYSUTIL table, and the DB2 and utility MEPLs.

```
DIAGNOSE
 DISPLAY SYSUTIL
DIAGNOSE
 DISPLAY MEPL
```

Example 5: Abend LOAD utility if message is issued. Abend the LOAD utility when unique index key violations occur.

```
DIAGNOSE
 ABEND MESSAGE U344
```

Example 6: Force dump if abend with specified reason code. Cause a dump to be taken if an abend occurs with either of the specified reason codes.

```
DIAGNOSE
 ALLDUMPS(X'00E40322',X'00E40323')
```

Chapter 2-9. LOAD

Use LOAD to load one or more tables of a table space. LOAD loads records into the tables and builds or extends any indexes 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 associated with the table, and any field procedure associated with any column of the table.

For a diagram of LOAD syntax and a description of available options, see “Syntax and options of the control statement” on page 122. For detailed guidance on running this utility, see “Instructions for running LOAD” on page 148.

Output: Output from LOAD DATA consists of one or more of the following:

- A loaded table space or partition
- A discard file of rejected records
- A summary report of errors encountered during processing, generated only if you specify ENFORCE CONSTRAINTS or if the LOAD involves unique indexes

Related information: For information regarding ESA data compression, see Section 2 (Volume 1) of *DB2 Administration Guide*.

Authorization required: To execute this utility, the privilege set of the process must include one of the following:

- Ownership of the table
- LOAD privilege for the database
- DBADM or DBCTRL authority for the database
- SYSCTRL or SYSADM authority

You cannot run the LOAD utility on the DSNDB01 or DSNDB06 databases, except to add lines to SYSIBM.SYSSTRINGS.

LOAD operates on a table space, so you must have authority for all tables in the table space when you perform LOAD.

To run LOAD STATISTICS REPORT YES, the privilege set must include the SELECT privilege on the catalog tables.

Execution phases of LOAD: Table 16 describes the phases of the LOAD utility.

Table 16 (Page 1 of 2). LOAD phases

Phase	Description
UTILINIT	Initialization and setup.

Table 16 (Page 2 of 2). LOAD phases

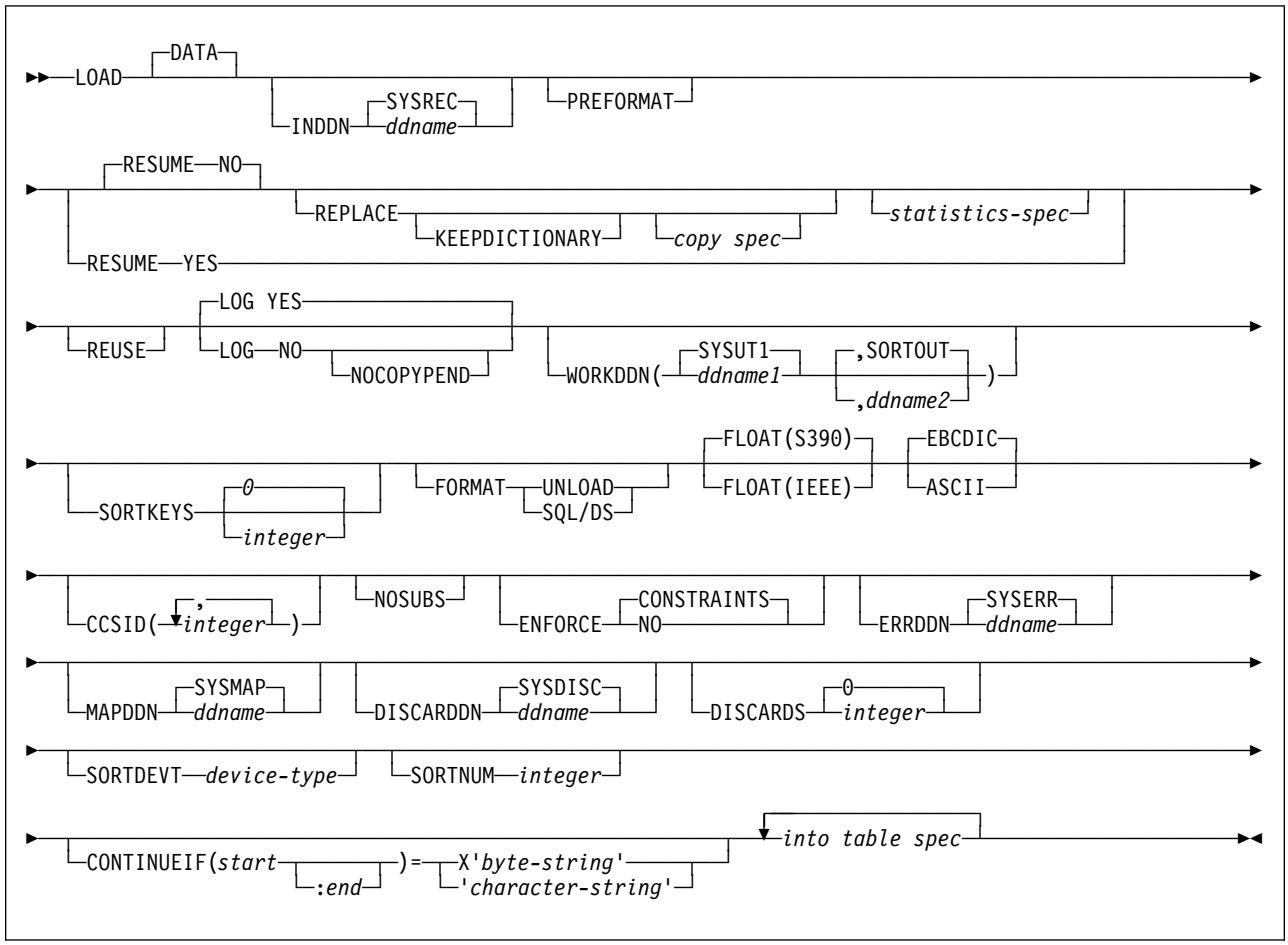
Phase	Description
RELOAD	Loading of record types and writing of temporary file records for indexes and foreign keys. Check constraints are checked for each row. One pass through the sequential input data set is made. Internal commits are taken to provide commit points at which to restart in case operation should halt in this phase. Creates inline copies if you specified the COPYDDN or RECOVERYDDN keywords. If SORTKEYS is used, a subtask is started at the beginning of the RELOAD phase to handle the work of sorting the keys. The sort subtask initializes and waits for the main RELOAD phase to pass its keys to SORT. The RELOAD phase 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. PREFORMAT for table spaces occurs at the end of the RELOAD phase.
SORT	Sorting of 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 processed during the RELOAD phase: • There is not more than one key per table • All keys are the same type (index key, index foreign key, nonindexed foreign key) • The data being loaded or reloaded is in key order (if a key exists) • The data being loaded or reloaded is grouped by table and each input record is loaded into one table only. If you use the SORTKEYS keyword, SORT passes the sorted keys in memory to the BUILD phase, which builds the indexes.
BUILD	Creating indexes from temporary file records for all indexes defined on the loaded tables. Detection of duplicate keys. Preformatting of indexes occurs at the end of the build phase.
SORTBLD	If you specify a parallel index build, all activities that normally occur in both the SORT and BUILD phases occur in the SORTBLD phase instead.
INDEXVAL	Correction of unique index violations from the information in SYSERR, if any exist.
ENFORCE	Checking of referential constraints, and correction of violations. Information about violations of referential constraints are stored in SYSERR.
DISCARD	Copying of records causing errors from the input data set to the discard data set.
REPORT	Generation of a summary report, if you specified ENFORCE CONSTRAINT or if load index validation is performed. The report is sent to SYSPRINT.
UTILTERM	Cleanup.

Syntax and options of the control statement

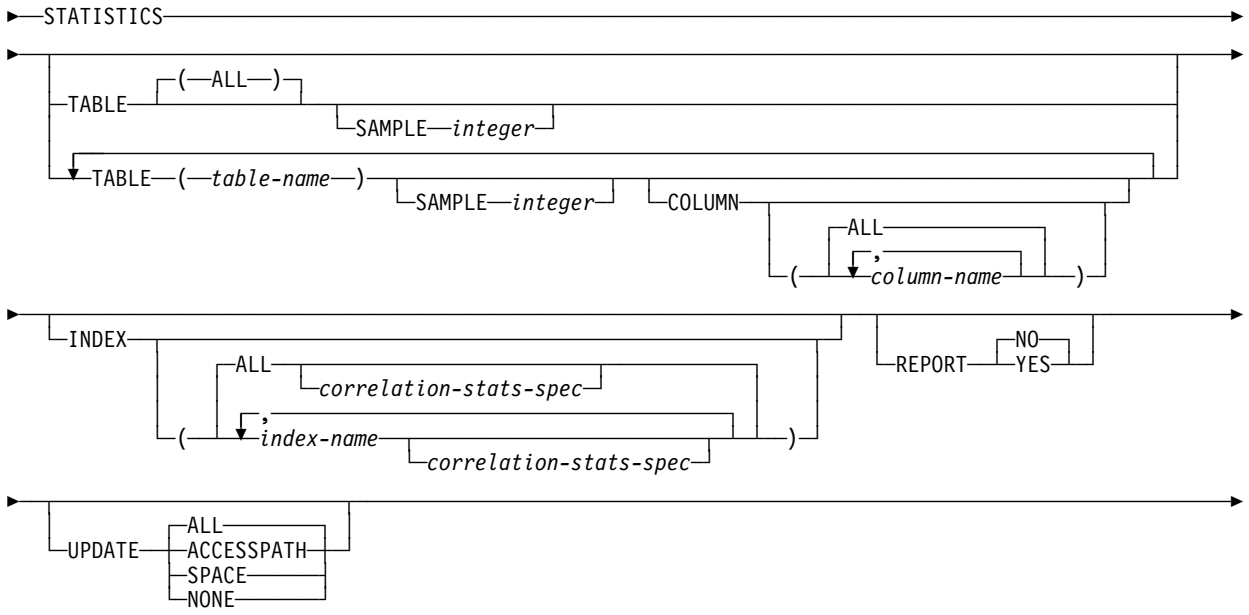
The utility control statement defines the function the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name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

Syntax diagram

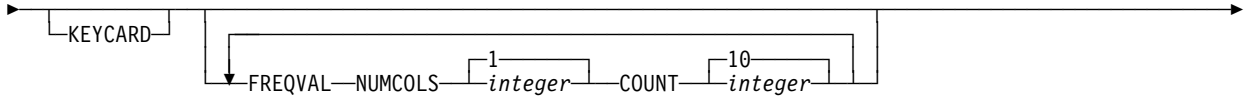
For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.



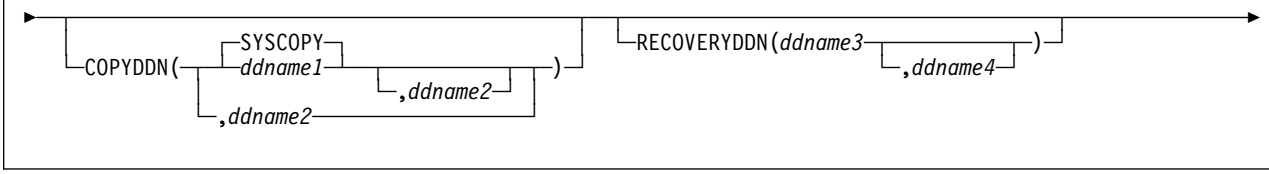
statistics-spec:



correlation-stats-spec:



copy spec:



Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see “Control statement coding rules” on page 27.

DATA Is used for clarity only. You identify the data selected for loading with *table-name* on the INTO TABLE option. See “INTO TABLE spec” on page 135 for a description of the statement.

INDDN *ddname*

Specifies the DD statement for the input data set. Record format for the input data set must be fixed or variable. The data set must be readable by the MVS BSAM access method.

ddname is the DD name.

The **default** is **SYSREC**.

PREFORMAT Specifies that the remaining pages are preformatted up to the high allocated RBA in the table space and index spaces associated with the table 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 its corresponding partitioning index space. 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 136 for the description for specifying PREFORMAT at the partition level.

RESUME Tells whether records are to be loaded into an empty or non-empty table space. For nonsegmented table spaces, space occupied by rows that have been marked as deleted or by rows of dropped tables is not reused.

Attention: Specifying LOAD RESUME (rather than PART *integer* RESUME) tells LOAD to drain the table space, which can inhibit concurrent processing of separate partitions. If you want to process other partitions concurrently, use "INTO TABLE spec" on page 135 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 containing 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 occupied by rows marked as deleted or by rows of dropped tables is not reused.

REPLACE Tells 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 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 using the REPLACE option or replace a single partition 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 description for specifying REPLACE at the partition level under the keyword descriptions for INTO TABLE.

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 associated with building a new dictionary.

This keyword is valid only if a compression dictionary exists and the table space being loaded has the COMPRESS YES attribute.

If the table space has the COMPRESS YES attribute, but there is no dictionary, one is built and a warning message is issued.

For information regarding ESA data compression, see Section 2 (Volume 1) of *DB2 Administration Guide*.

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.

The COPYDDN keyword can only be specified with REPLACE. A full image copy data set (SHRLEVEL REFERENCE) is created for the table or partitions specified when LOAD executes. The table space or partitions for which an image copy is produced is not placed in COPY pending status.

Image copies 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 will not create a copy of any index or LOB table space. You must perform these tasks separately.

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.

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.

TABLE Specifies the table for which column information is to be gathered. All tables must belong to the table space specified in the TABLESPACE option.

(ALL) Specifies that information is to be gathered for all columns of all tables in the table space.

SAMPLE *integer*
Indicates the percentage of rows to sample when collecting non-indexed column statistics. Any value from 1 through 100 can be specified. The default is *25*.

(*table-name***)** Specifies the tables for which column information is to be gathered. The parentheses are required. If you omit the qualifier, the user identifier for the utility job is used.

If you specify more than one table, you must repeat the TABLE option.

COLUMN Specifies columns for which column information is to be gathered.

You can only specify this option if you specify a 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.

(*column-name***, ...)**
Specifies the columns for which statistics are to be gathered. The parentheses are required.

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 specified in the TABLESPACE option.

(ALL) Specifies that the column information is to be gathered for all indexes defined on tables contained in the table space. The parentheses are required.

(*index-name***)** Specifies the indexes for which information is to be gathered. The parentheses are required.

REPORT Determines if a set of messages is generated to report the collected statistics.

NO Indicates that the set of messages is not output to SYSPRINT.

The **default** is **REPORT NO**.

YES Indicates that the set of messages is output to SYSPRINT. The messages generated are dependent on the combination of keywords (such as TABLESPACE, INDEX, TABLE, and COLUMN) 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 Tells whether the collected statistics are inserted into the catalog tables. UPDATE also allows you to select statistics used for access path selection or statistics used by database administrators.

ALL Indicates that all collected statistics will be updated in the catalog.

The **default** is **UPDATE ALL**.

ACCESSPATH Indicates that only the catalog table columns that provide statistics used for access path selection are updated.

SPACE Indicates that only the catalog table columns that provide statistics to help the database administrator assess the status of a particular table space or index are updated.

NONE Indicates that no catalog tables are updated with the collected statistics. This option is only valid when REPORT YES is specified.

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.

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 to concatenate together when collecting frequent values from the specified index. Specifying '3' means to collect frequent values on the concatenation of the first three key columns. The default is 1, which means collect frequent values on the first key column of the index.

COUNT Indicates the number of frequent values to be collected. Specifying '15' means collect 15 frequent values from the specified key columns. The default is *10*.

REUSE When used with the REPLACE option, specifies 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.

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 135 for information about specifying REUSE for individual partitions.

If a data set has multiple extents, the extents will not be released if you specify the REUSE parameter.

LOG Tells whether logging is to occur during the RELOAD phase of the load process.

YES Specifies normal logging during the load process. All records 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 until the restriction is removed. For ways to remove the restriction, see “Resetting the COPY pending status” on page 174.

If you load a single partition of a partitioned table space and the table space has a nonpartitioning index, some logging might occur in the build phase as DB2 logs index structure changes. This logging allows recoverability of the nonpartitioning index in case an abend occurs, and also allows concurrency.

A LOB table space that was defined with LOG YES or LOG NO will affect logging while loading a LOB column. See Table 25 on page 170 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 will not turn on or change any informational COPY pending (ICOPY) status for indexes. Normal completion of a LOAD LOG NO NOCOPYPEND job will be return code 0 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: You should only specify the NOCOPYPEND option if the data in the table space can be easily recreated by another LOAD if it is lost. If you do not take an image copy following the LOAD, you will not be able to recover the table space using the RECOVER utility.

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**.

SORTKEYS *integer*
Specifies that index keys will be sorted in parallel during the SORTBLD phase to improve performance. Optionally, you may use *integer* to provide an estimate of the number of index keys to be sorted. The default is **0**.

SORTKEYS is recommended to improve performance unless the table space has no indexes, or it has only one index and the data being loaded is already sorted in key sequence. For more information about invoking a parallel index build, see “Improving performance with SORTKEYS” on page 161 and “Building indexes in parallel for LOAD” on page 165.

FORMAT Identifies the format of the input record. If you use FORMAT, it uniquely determines the format of the input, and no field specifications are allowed in an INTO TABLE option. Follow FORMAT with either the UNLOAD or SQL/DS option.

If you omit FORMAT, the format of the input data is determined by the rules for field specifications described for the WHEN option of “Option descriptions for INTO TABLE” on page 136.

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 there is an INTO TABLE option to specify each table. Do not add columns or change column definitions of tables between running REORG UNLOAD ONLY and LOAD FORMAT UNLOAD. Any WHEN clause on that statement is ignored; DB2 reloads the records into the same tables from which they were unloaded. This 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 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 tells which input records load into which DB2 table.

For information on the correct DCB parameters to specify on the DD statement for the input data set, refer to *DB2 Server for VSE: DBS Utility* or *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 for OS/390” on page 513.

SQL/DS data that has been unloaded to disk under DB2 Server for VSE & VM resides in a simulated OS/390 type data set with a record format of VBS. That must be taken into consideration when transferring the data to another system to be loaded into a DB2 table (for example, the DB2 Server for VSE & VM FILEDEF must define it as an OS/390 type data set). Processing it 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.

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, and is 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 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. For more information about the BFP format, see *ESA/390 Principles of Operation*.

EBCDIC Specifies that the input data file is EBCDIC. **EBCDIC** is the default.

ASCII Specifies that the input data file is ASCII. Numeric, date, time, and timestamp internal formats are not affected by the ASCII option.

CCSID Specifies up to three coded character set identifiers (CCSIDs) for the input file. The first specifies the CCSID for SBCS data found in the input file, the second specifies the CCSID for mixed DBCS data, and the third specifies the CCSID for DBCS data. If any of these are 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; that is, if the input data is EBCDIC, the omitted CCSIDs are assumed to be the EBCDIC CCSIDs specified at installation, and if the input data is ASCII, the omitted CCSIDs are assumed to be the ASCII CCSIDs specified at installation. If the CCSIDs of the input data file do not match the CCSIDs of the table being loaded, the input data is converted to the table CCSIDs before being loaded.

integer is any valid CCSID specification.

NOSUBS Specifies that LOAD is not to accept substitution characters in a string.

A substitution character is sometimes placed in a string when that string is being converted from ASCII to EBCDIC, or converted from one CCSID to another. For example, this substitution occurs when a character (sometimes referred to as a codepoint) 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 or not LOAD is to enforce check constraints and referential constraints.

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 are required.

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 or check constraint is defined for the table.

ERRDDN *ddname*

Specifies the DD statement for a work data set for error processing. Information about errors 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**.

MAPDDN *ddname*

Specifies the DD statement for a work data set for error processing. It is used to map the identifier of a table row back to 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**.

DISCARDDN *ddname*

Specifies the DD statement for a "discard data set," 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 loaded, then removed (due to unique index errors, or referential or check constraint violations). Input records can be flagged 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.

If you omit the DISCARDDN option, the utility application program saves discarded records only if there is a SYSDISC DD statement in the JCL input.

The **default** is **SYSDISC**.

DISCARDS *integer*

Specifies the maximum number of source records to be written on the discard data set. *integer* can range from 0 to 2147483647. If the discard maximum is reached, LOAD abends, 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 there is no maximum. The entire input data set can be discarded.

The **default** is **DISCARDS 0**.

SORTDEVT *device-type*

Specifies the device type for temporary data sets to be dynamically allocated by DFSORT. It can be any device type acceptable to the DYNALLOC parameter of the SORT or OPTION options for DFSORT.

If you omit SORTDEVT and a sort is required, you must provide the DD statements that the sort application program needs for the temporary data sets.

SORTNUM *integer*

Tells the number of temporary data sets to be dynamically allocated by the sort application program.

If you omit SORTDEVT, SORTNUM is ignored. If you use SORTDEVT and omit SORTNUM, no value is passed to DFSORT. It is allowed to take its own default.

CONTINUEIF Allows you to treat each input record as a portion of a larger record. After CONTINUEIF, write a condition in one of these 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, but character constants specified in the utility control statement are always interpreted as EBCDIC. To use CONTINUEIF when the ASCII option is specified, code the condition using the hexadecimal form, not the character string form. For example, use `(1:1)=X'31'` rather than `(1:1)='1'`.

(*start:end***)** Are 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 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*, the length of the continuation field is taken as 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 one is padded. Character strings are padded with blanks. Hexadecimal strings are padded with zeros.

X'*byte-string***'**

Is a string of hexadecimal characters. That value in the continuation field indicates that the next input record is a continuation of the current load record. Records with that value are concatenated until the value in the continuation field changes. For example, a specification might be

```
CONTINUEIF (72) = X'FF'
```

'*character-string***'**

Is a string of characters that has the same effect as X'*byte-string*'. For example, a specification might be

```
CONTINUEIF (99:100) = 'CC'
```

INTO TABLE spec

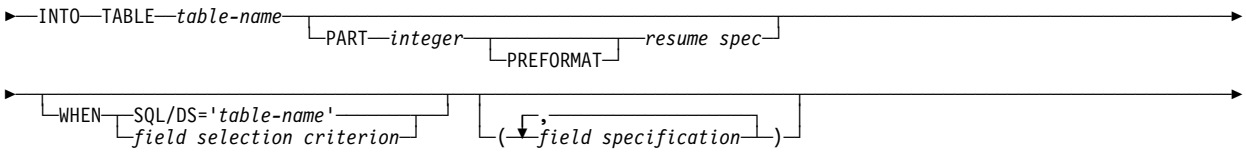
More than one table or partition for each table space can be loaded with a single invocation of the LOAD utility. At least one INTO TABLE statement is required for each table to be loaded, to:

- Identify the table that is to be loaded
- Describe fields within the input record
- Define the format of the input data set.

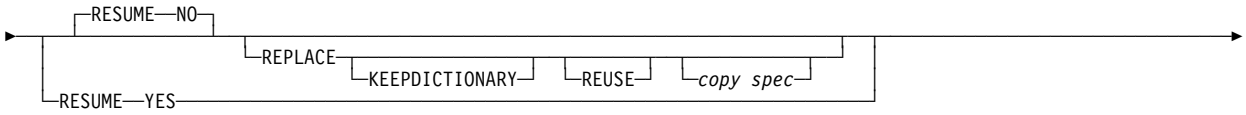
All tables specified by INTO TABLE statements must belong to the same table space.

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.

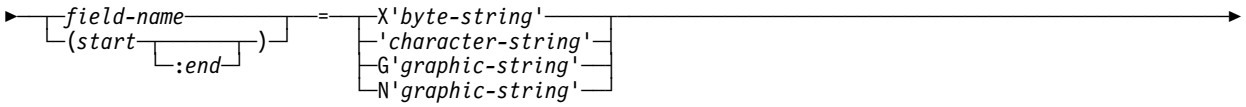
INTO TABLE spec:

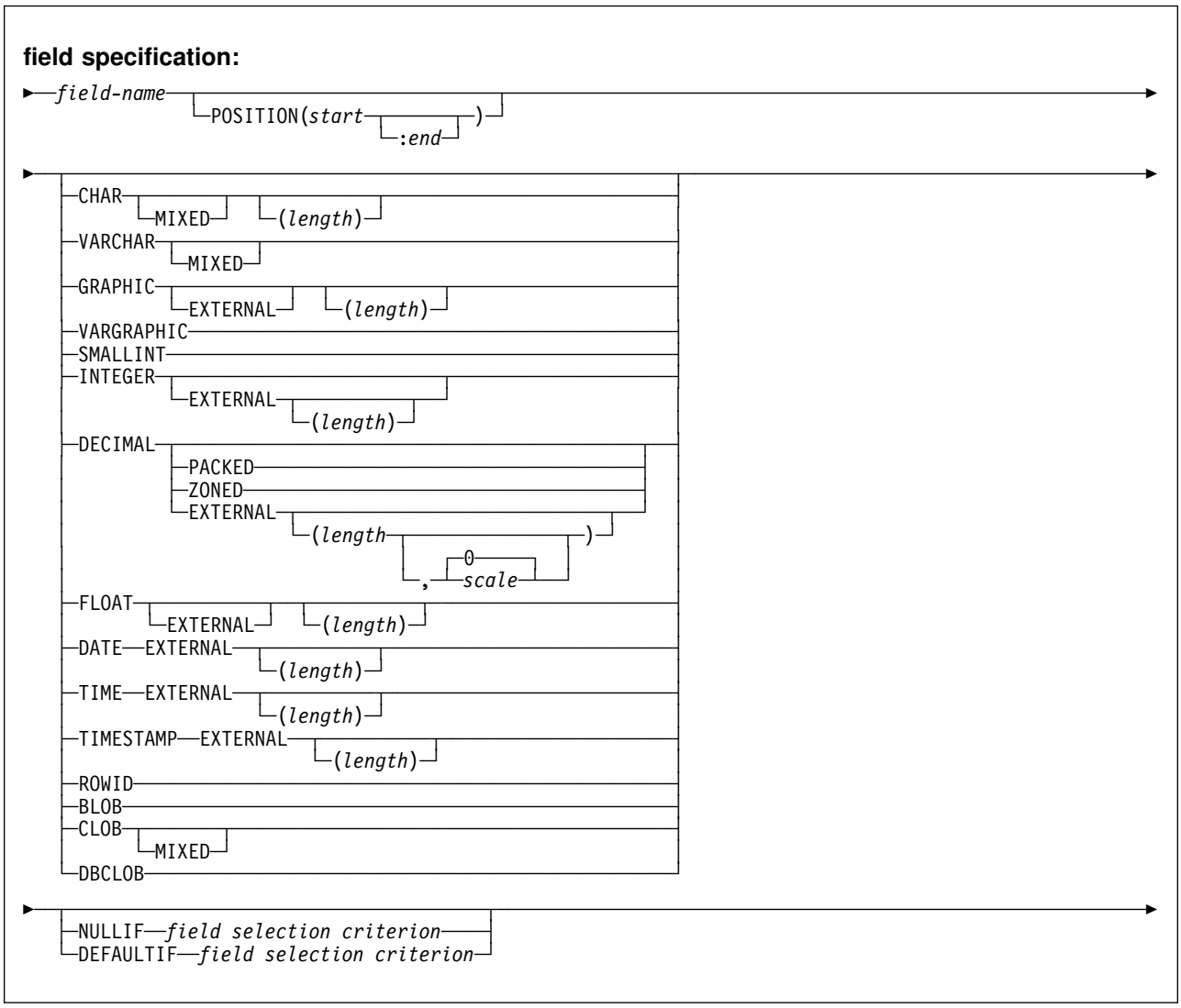


resume spec:



field selection criterion:





Option descriptions for INTO TABLE

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see "Control statement coding rules" on page 27.

table-name Is the name of a table to be loaded. 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 invoker of the utility job step is used as the qualifier of the table name.

Data from every load record in the data set is loaded into the table specified 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 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 not generated by data conversion occur

The following keywords are optional:

PART *integer* Is valid only for partitioned table spaces.

integer is the number of the partition for which records are accepted for loading. Any data outside the range of the specified partition is not loaded. The maximum is 254.

LOAD INTO PART *integer* is not allowed if an identity column is
part of the partitioning index.

PREFORMAT Specifies that the remaining pages are preformatted up to the high allocated RBA in the partition and its corresponding partitioning index space. The preformatting occurs after the data has been 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 occupied by 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 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 containing 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 If specified, this option indicates that you want to replace only the contents of the partition 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.

KEEPDICTIONARY
Prevents the LOAD utility from building a new dictionary. LOAD retains the current dictionary and uses it for compressing the input data. This option eliminates the cost associated with building a new dictionary.

This keyword is only valid if a dictionary exists and the partition being loaded has the COMPRESS YES attribute.

If the partition has the COMPRESS YES attribute, but there is no dictionary, one is built and an error message is issued.

REUSE When used with the REPLACE option, specifies 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 the "table space level" LOAD) only the specified partitions are logically reset. If you specify REUSE for the table space and REPLACE for the partition, then data sets for the replaced parts are logically reset.

WHEN The WHEN clause tells which records in the input data set are to be loaded. If there is no WHEN clause (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 beyond the range of the partition specified 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, but character constants specified in the utility control statement are always interpreted as EBCDIC. To use WHEN when the ASCII option is specified, code the condition using the hexadecimal form, not the character string form. For example, use (1:1)=X'31' rather than (1:1)='1'.

SQL/DS='*table-name***'**
Is valid only when the FORMAT SQL/DS option is used on the LOAD statement.

table-name is the SQL/DS name of a table that has been unloaded onto the SQL/DS unload data set. The table name after INTO TABLE tells which DB2 table the SQL/DS table is loaded into.

If there is no WHEN clause, input records from every SQL/DS table are loaded into the table 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 loaded into the table 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—stored with length fields—that *precede* the selection field, then they must be padded so the start of the selection field is always at the same offset.

The field and the constant need not 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
: Is the name of a field 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)
: *start* and *:end* are column numbers in the assembled load record; the first column of the record is column 1. The two numbers tell 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***'**
: Gives the constant as a string of hexadecimal characters. For example, write

```
WHEN (33:34) = X'FFFF'
```

'*character-string***'**
: Gives the constant as a string of characters. For example, write

```
WHEN DEPTNO = 'D11'
```

G'*graphic-string***'**
: Gives the constant as a string of double-byte characters. For example, write

```
WHEN (33:36) = G'<**>'
```

  where “<” 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. You can specify G either as an upper- or lower-case character.

N'*graphic-string***'**
Gives the constant as a string of double-byte characters. N and G are synonymous for specifying graphic string constants. You can specify N either as an upper- or lower-case character.

(*field-specification***, ...)**
Describes the location, format, and null value identifier of the data 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 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; there cannot be intervening gaps between them and the fields that follow.
- ROWID fields are varying length, and must contain a valid 2-byte binary length field preceding the data; there cannot be intervening gaps between them and the fields that follow.
- LOB data types are varying length, and require a valid 4-byte binary length field preceding the data; there cannot be intervening gaps between them and the 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, there must be a field specification 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 may appear in the field specification only if they were defined with the GENERATED BY DEFAULT attribute.

field-name
Is the name of a field, and 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 named after INTO TABLE. The field-name can be used as a vehicle to specify the range of incoming data. See page 180.

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.

The length of the field is determined 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 preceding the data. For VARCHAR fields, the length is in bytes; for VARGRAPHIC fields, it is in (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 preceding the data. For BLOB and CLOB fields, the length is in bytes; for DBCLOB fields, it is in (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 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 17 on page 142 shows the default length, in bytes, for each data type.

Table 17. Default length of each data type (in bytes)

Data Type	Length in Bytes
BLOB	Varying
CHARACTER	Length used in column definition
CLOB	Varying
DATE	10 (or installation default)
DBCLOB	Varying
DECIMAL EXTERNAL	Same as for DECIMAL ZONED
DECIMAL PACKED	Length implied by column definition
DECIMAL ZONED	Decimal precision if decimal output column, otherwise length implied by column definition
FLOAT (single precision)	4
FLOAT (double precision)	8
GRAPHIC	2 * (length used in column definition)
INTEGER	4
MIXED	Mixed DBCS data
ROWID	Varying
SMALLINT	2
TIME	8 (or installation default)
TIMESTAMP	26
VARCHAR	Varying
VARGRAPHIC	Varying

If a data type is not given for a field, its data type is taken to be the same as that of the column it is loaded into, as given in the DB2 table definition.

POSITION(*start:end***)**

Tells 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 given 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 specified by * .

The POSITION option specification *cannot* be enclosed in parentheses; however, the *start:end* description *must* be enclosed in parentheses, as the following example shows:

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 designated EXTERNAL are in the same format in the input records.

CHAR
CHAR(*length***)** For a fixed-length character string. The length of the string is determined from the POSITION specification or from *length*. You can also specify CHARACTER and CHARACTER(length).

MIXED Specifies that the input field contains mixed SBCS and DBCS data. If MIXED is specified, then any required CCSID conversions use the mixed CCSID for the input data; if MIXED is not specified, then any such conversions will use the SBCS CCSID for the input data.

VARCHAR For a character field of varying-length. The length in bytes must be given in a 2-byte binary field preceding the data. (The length given there does not include the 2 byte field itself.) The length field must start in the column specified as *start* in the POSITION option. If *:end* is used, it is ignored.

MIXED Specifies that the input field contains mixed DBCS data. If MIXED is specified, then any required CCSID conversions use the mixed CCSID for the input data; if MIXED is not specified, then any such conversions will use the SBCS CCSID for the input data.

GRAPHIC EXTERNAL
GRAPHIC EXTERNAL(*length***)**
Used for a graphic field. 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. Aside from the shift characters, there must be an even number of bytes in the field. The first byte of any pair must not be a shift character.

length is a number of double-byte characters. *length* for GRAPHIC EXTERNAL does not include the bytes of shift characters. The length of the field in bytes is twice the value of *length*.

For example, let *** represent 3 double-byte characters, and let < and > represent shift-out and shift-in characters. Then, to describe <***>, use either POS(1:8) GRAPHIC EXTERNAL or POS(1) GRAPHIC EXTERNAL(3).

GRAPHIC
GRAPHIC(*length***)**
Used for a graphic field. 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 a number of double-byte characters. The length of the field in bytes is twice the value of *length*.

For example, let *** represent 3 double-byte characters. Then, to describe ***, use either POS(1:6) GRAPHIC or POS(1) GRAPHIC(3). A GRAPHIC field described in this way cannot be specified in a field selection criterion.

VARGRAPHIC

For a graphic field of varying-length. The length, in double-byte characters, must be given in a 2-byte binary field preceding the data. (The length given there does not include the 2 byte field itself.) The length field must start in the column specified as *start* in the POSITION option. *:end*, if used, is ignored.

VARGRAPHIC input data must not contain shift characters.

SMALLINT For a two-byte binary number. Negative numbers are in two's complement notation.

INTEGER Specifies a four-byte binary number. Negative numbers are in two's complement notation.

You can also specify INT.

INTEGER EXTERNAL
INTEGER EXTERNAL(*length***)**

A string of characters that represent a number. The format is that of an SQL numeric constant as described in Chapter 3 of *DB2 SQL Reference*.

You can also specify INT EXTERNAL.

DECIMAL
DECIMAL PACKED

For a number of the form *ddd...ds*, where *d* is a decimal digit represented by four bits, and *s* is a four-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 allowed in the SQL definition.

You can also specify DEC or DEC PACKED.

DECIMAL ZONED

For a number of the form *znznzn...z/sn*, where *n* is a decimal digit represented by the right four bits of a byte (called the *numeric bits*); *z* is that digit's zone, represented by the left four bits; and *s* is the rightmost byte of the decimal operand, and 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 allowed in the SQL definition.

You can also specify DEC ZONED.

DECIMAL EXTERNAL
DECIMAL EXTERNAL(*length***)**
DECIMAL EXTERNAL(*length,scale***)**

A string of characters that represent a number. The format is that of an SQL numeric constant as described in Chapter 3 of *DB2 SQL Reference*.

length Overall length of the input field in bytes.

scale Specifies the number of digits to the right of the decimal point. That number must be an integer greater than or equal to 0, and can be greater than *length*.

The **default** is **0**.

If scale is greater than length, or the number of digits provided is less than the scale specified, 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 target scale are truncated. If scale is specified and the target column is 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.

You can also specify DEC EXTERNAL and DEC EXTERNAL(*length*).

FLOAT EXTERNAL
FLOAT EXTERNAL(*length***)**

A string of characters that represent a number. The format is that of an SQL floating point constant as described in Chapter 3 of *DB2 SQL Reference*.

If you specified the FLOAT(IEEE) or FLOAT(S390) option, it does not apply for this format (string of characters) of floating point numbers.

FLOAT(*length***)**

For 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 in excess-64 notation
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 in excess-64 notation
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 in excess-64 notation
Bits 12-63 Represent a mantissa.

You can also specify REAL for single precision floating point and DOUBLE PRECISION for double precision floating point.

DATE EXTERNAL
DATE EXTERNAL(*length***)**

For a character string representation of a date. Length, if unspecified, is the length given by the LOCAL DATA LENGTH install option, or, if none was provided, defaults to 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. Trailing blanks can be included, but no leading blanks are allowed.

- *dd.mm.yyyy*
- *mm/dd/yyyy*
- *yyyy-mm-dd*
- any local format defined by your site at the time DB2 was installed.

TIME EXTERNAL
TIME EXTERNAL(*length***)**

For a character string representation of a time. Length, if unspecified, is the length given by the LOCAL TIME LENGTH install option, or, if none was provided, defaults to eight 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 defined by your site 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
TIMESTAMP EXTERNAL(*length***)**

For a character string representation of a time. *length*, if unspecified, defaults to 26 bytes. If you specify a length, it must be within the range of 19 to 26 bytes.

Timestamps can be in either 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*

See Chapter 3 of *DB2 SQL Reference* for more information about the timestamp data type.

ROWID For a row ID. The input data must be a valid value for a row ID; DB2 will not perform any conversions.

A field specification for a row ID column is not allowed if the row ID column was created GENERATED ALWAYS.

If the ROWID column is part of the partitioning key, LOAD INTO TABLE PART is not allowed; specify LOAD INTO TABLE instead.

BLOB For a BLOB field. You must specify the length in bytes in a 4-byte binary field preceding the data. (The length does *not* include the 4-byte field itself.) The length field must start in the column specified as *start* in the POSITION option. If *:end* is used, it is ignored.

CLOB For a CLOB field. You must specify the length in bytes in a 4-byte binary field preceding the data. (The length does *not* include the 4-byte field itself.) The length field must start in the column 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, then any required CCSID conversions use the mixed CCSID for the input data; if MIXED is not specified, then any such conversions will use the SBCS CCSID for the input data.

DBCLOB For a DBCLOB field. You must specify the length in double-byte characters in a 4-byte binary field preceding the data. (The length does *not* include the 4-byte field itself.) The length field must start in the column specified as *start* in the POSITION option. If *:end* is used, it is ignored.

Field selection criterion: The criterion describes a condition that causes the DB2 column to be loaded with NULL or its default value.

NULLIF *field-selection-criterion*

Describes a condition that causes the DB2 column to be loaded with NULL. The *field-selection-criterion* can be written with the same options as described on page 138. If the contents of the NULLIF field match the character constant given, the field specified in *field-specification* is loaded with NULL.

If the NULLIF field is defined by the name of a VARCHAR or VARGRAPHIC field, the length of the field is taken 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, but character constants specified in the utility control statement are always interpreted as EBCDIC. To use NULLIF when the ASCII option is specified, code

the condition using the hexadecimal form, not the character string form. For example, use (1:1)=X'31' rather than (1:1)='1'.

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, with the field specification:

```
(FIELD1 POSITION(*) CHAR(4)
 FIELD2 POSITION(*) CHAR(3) NULLIF(FIELD1='SKIP')
 FIELD3 POSITION(*) CHAR(5))
```

and the source record:

```
SKIP   FLD03
```

the record is loaded so that:

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.

DEFAULTIF *field-selection-criterion*

Describes a condition that causes the DB2 column to be loaded with its default value. The *field-selection-criterion* can be written with the same options as described on page 138. If the contents of the DEFAULTIF field match the character constant given, the field 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, the length of the field is taken 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, but character constants specified in the utility control statement are always interpreted as EBCDIC. To use DEFAULTIF when the ASCII option is specified, code the condition using the hexadecimal form, not the character string form. For example, use `(1:1)=X'31'` rather than `(1:1)='1'`.

| You can use the DEFAULTIF attribute with the ROWID keyword. If
| the condition is met, the column will be loaded with a value
| generated by DB2.

Instructions for running LOAD

To run LOAD, you must:

1. Read “Before running LOAD” on page 149 in this chapter.
2. Prepare the necessary data sets, as described in “Data sets used by LOAD” on page 149.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27. (For examples of JCL for LOAD, see “Sample control statements” on page 178.)

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 152. (For a complete description of the syntax and options for LOAD, see “Syntax and options of the control statement” on page 122.)
5. Check the compatibility table in “Concurrency and compatibility” on page 173 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 170.
7. Read “After running LOAD” on page 174 in this chapter.
8. Run LOAD.

See “Chapter 2-1. Invoking DB2 online utilities” on page 27 for an explanation of ways to execute DB2 utilities.

Before running LOAD

Preprocessing input data: There is no sorting of the data rows during the LOAD utility— rows are loaded in the physical sequence in which they are found. It is a good idea to sort your input records in clustering sequence before loading.

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.

When loading into a segmented table space, sort your data by table to ensure that the data is loaded in the best physical organization.

Data sets used by LOAD

Table 18 describes the data sets used by LOAD. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 18 (Page 1 of 2). Data sets used by LOAD

Data Set	Description	Required?
SYSIN	Input data set containing the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
Input data set	The input data set containing the data to be loaded. Its name is identified by the DD statement specified by the INDDN option. The default name is SYSREC. It must be a sequential data set that is readable by BSAM.	Yes
Sort data sets	Two temporary work data sets for sort input and sort output. Their DD names are specified 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[1,3,5]

Table 18 (Page 2 of 2). Data sets used by LOAD

Data Set	Description	Required?
Mapping data set	Work data set for mapping the identifier of a table row back to the input record that caused an error. The default DD name is SYSMAP.	Yes[1,4]
UTPRINT	Contains messages from DFSORT (usually, SYSOUT or DUMMY).	No[6]
Discard data set	A work data set to hold copies of records not loaded. It must be a sequential data set that is readable by BSAM. Its name is specified with the DISCARDDN option of the utility control statement. If you omit this, LOAD creates the data set with the same record format, record length, and block size as the input data set. The default DD name is SYSDISC.	Yes[2]
Error data set	Work data set for error processing. Its DD name is specified with the ERRDDN parameter of the utility control statement. The default DD name is SYSERR.	Yes
Copy data sets	1 to 4 output data sets to contain image copy data sets. Their DD names are specified with the COPYDDN and RECOVERYDDN options of the utility control statement.	No

Note:

[1] When referential constraints exist and ENFORCE(CONSTRAINTS) is specified.
[2] If you request discard processing, by using the DISCARDS option of the utility control statement.
[3] For tables with indexes.
[4] If you request discard processing when loading one or more tables that have unique indexes.
[5] If SORTKEYS is specified with no estimate or an estimate of 0.
[6] Required if a sort is done.

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

Table The name of the table to be loaded. It is named in the LOAD control statement and is accessed through the DB2 catalog. (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 Table 19 on page 151 to calculate the size of work data sets for LOAD.

Table 19. Work data set calculation

Work Data Set	Size
SYSUT1	• Simple table space: *max(k,e)* • Partitioned or segmented table space: *max(k,e,m)* If SORTKEYS is used and an estimate of the number of keys to be sorted is specified: *max(f,e)* for a simple table space *max(f,e,m)* for a partitioned or segmented table space.
SORTOUT	*max(k,e)* If SORTKEYS is used, *max(f,e)*
SYSERR	*e*
SYSMAP	• Simple table space or discard processing: *m* • Partitioned or segmented table space without discard processing: *max(m,e)*
SYSDISC	Same size as input data set

Notes:

1.

k = key calculation
f = foreign key calculation
m = map calculation
e = error calculation
max() = maximum value of the specified calculations

Calculating the key: k

max(longest index key + 14, longest foreign key + 14) × (number of keys extracted)

Calculating the number of keys extracted:

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.
   b. Count 1 for subsequent relationships in which the foreign key participates (if any).
4. Multiply count by the number of rows to be loaded.

Calculating the foreign key: f

max(longest foreign key + 14) × (number of keys extracted)

Calculating the map: m

The data set must be large enough to accommodate 1 map entry (length = 21 bytes) per table row produced by the LOAD job.

Calculating the error: e

The data set must be large enough to accommodate 1 error entry (length = 100 bytes) per defect 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, the number of possible defects can be calculated as follows:

  number of input records +
  (number of unique indexes × *number of keys extracted)* +
  (number of relationships × *number of foreign keys extracted)*

- For nondiscard processing, the data set is not required.

Allocating twice the space 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 further information, 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 you load the data into. 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 you have a variable-length column containing 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' for six single-byte characters in a VARCHAR column
- **X'0003'**X'42C142C142C2' for three double-byte characters in a VARGRAPHIC column

Ordering loaded records

LOAD 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 does not insert records in sequence with existing records, even if there is a clustering index. 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 afterwards.

As rows with duplicate key values for unique indexes fail to be loaded, any records dependent on such rows will either:

- Fail to be loaded because they would cause referential integrity violations (if you specify ENFORCE CONSTRAINTS)
- Be loaded without regard to referential integrity violations (if you specify ENFORCE NO).

This might mean violations of referential integrity. 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 single-table table space or in a multiple-table table space. You can replace all the data in a table space (using the REPLACE option), or you can load new records into a table space without destroying the rows already there (using the RESUME option).

| If an object is in REORG pending status, you can perform a LOAD REPLACE of
| the entire table space (which resets REORG pending status). In this situation, no
| other LOAD operations are allowed. See Appendix C, “Resetting an advisory or
| restrictive status” on page 527 for more information.

Using LOAD REPLACE with LOG YES: LOAD REPLACE or PART REPLACE with LOG YES 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: Figure 7 is an example that replaces one table in a single-table table space:

```
LOAD DATA
  REPLACE
  INTO TABLE DSN8610.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 7. 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 have to work with one table at a time, 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:

1. Use LOAD REPLACE on the first table. This empties out the table space and replaces just the data for the first table.

```
LOAD DATA CONTINUEIF(72:72)='X'
  REPLACE
  INTO DSN8610.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 8. Example of using LOAD REPLACE on the first table in a table space

2. Use LOAD with RESUME YES on the second table.

```
LOAD DATA CONTINUEIF(72:72)='X'
  RESUME YES
  INTO DSN8610.TDSPTXT
  ( DSPINDEX  POSITION (2)     CHAR(2),
    LINENO    POSITION (6)     CHAR(2),
    DSPLINE   POSITION (80)    CHAR(79) )
```

Figure 9. Example of using LOAD with RESUME YES on the second table in a table space

   This adds the records for the second table without destroying the data in the first table.

If you need to replace just one table in a multi-table table space, you need to delete all the rows in the table, then use LOAD with RESUME YES. For example, assume you want to replace all the data in DSN8610.TDSPTXT without changing any data in DSN8610.TOPTVAL. To do this:

1. Delete all the rows from DSN8610.TDSPTXT using an SQL DELETE statement. (The mass delete works most quickly on a segmented table space.)

   ```
   DELETE FROM DSN8610.TDSPTXT;
   ```

2. Use the LOAD job in Figure 10 to replace the rows in that table.

```
LOAD DATA CONTINUEIF(72:72)='X'
  RESUME YES
  INTO DSN8610.TDSPTXT
  ( DSPINDEX  POSITION (2)     CHAR(2),
    LINENO    POSITION (6)     CHAR(2),
    DSPLINE   POSITION (80)    CHAR(79) )
```

Figure 10. Example of using LOAD with RESUME YES to replace one table in a multi-table table space

Adding more data to a table or partition

You may 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, the input data set should be specified in the JCL as DD DUMMY. LOAD REPLACE is efficient because:

1. LOAD REPLACE does not log any rows.
2. LOAD REPLACE redefines the table space.
3. LOAD REPLACE retains all views and privileges associated with a table space or table.
4. LOG YES can be used to make the LOAD REPLACE recoverable.

LOAD REPLACE will replace 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 following example loads 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, simplified to illustrate the point, does not list field specifications for all columns of the table.)

Attention: 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.

```
LOAD DATA CONTINUEIF(72:72)='X'
  INTO TABLE DSN8610.EMP PART 1 REPLACE WHEN (1) = '0'
  ( EMPNO     POSITION (1:6)  CHAR(6),
    FIRSTNME  POSITION (7:18) CHAR(12),

  ⋮
                                       )
  INTO TABLE DSN8610.EMP PART 2 RESUME YES WHEN (1) = '1'
  ( EMPNO     POSITION (1:6)  CHAR(6),
    FIRSTNME  POSITION (7:18) CHAR(12),

  ⋮
                                       )
```

The following example assumes you have your data in separate input data sets. That data is already sorted by partition, so you do not have to use the WHEN clause of INTO TABLE. The RESUME YES option placed 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 DSN8610.EMP REPLACE PART 1

LOAD DATA INDDN EMPLDS2 CONTINUEIF(72:72)='X'
  RESUME YES
  INTO TABLE DSN8610.EMP REPLACE PART 2
```

The following example allows partitioning independence when loading more than one partition concurrently.

```
LOAD DATA INDDN SYSREC LOG NO
  INTO TABLE DSN8610.EMP PART 2 REPLACE
```

LOAD INTO PART *x* is not allowed if an identity column is part of the partitioning index.

Loading data with referential constraints

If you plan to let DB2 enforce referential integrity in a set of tables, then you should already have read the section on implications for utility operations in Section 2 (Volume 1) of *DB2 Administration Guide*.

LOAD does not load a table with an incomplete definition; if the table has a primary key, then the unique index on that key must exist. If any table named to be loaded has an incomplete definition, the LOAD job terminates.

By default, LOAD enforces referential constraints. By doing that, it provides you with several possibilities for error:

- Records to be loaded might have duplicate values of a primary key.
- Records 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 sections 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, it checks their validity with respect to the constraints, identifies any invalid record by an error message, and deletes the record. At your choice, the record can also be copied 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.

But the project table has a primary key, the project number. In this case, the record rejected by LOAD defines a project number, and any record 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, both types of error records are copied to it.

Missing primary key values: The deletion of invalid records does not cascade to other dependent tables 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. Further, 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. Records referencing that department number might already exist in the project activity table. LOAD, therefore, automatically places the table space containing the project activity table (and all table spaces containing dependent tables of any table 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 records that violate a referential constraint. There are 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 175.

Consequences of ENFORCE NO: If you use the ENFORCE NO option, you tell LOAD not to enforce referential constraints. Sometimes there are good reasons for doing that (see Section 2 (Volume 1) of *DB2 Administration Guide*). 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 containing 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 spaces.

Correcting referential constraint violations

The referential integrity checking in LOAD can only delete incorrect dependent rows, which were input to LOAD. Deletion is not always the best strategy for correcting referential integrity violations.

For example, the violations may occur because parent rows do not exist. In this case, it is better to correct the parent table, not to delete the dependent rows. Therefore and in this case, ENFORCE NO would be more appropriate than ENFORCE CONSTRAINTS. After the parent table is corrected, CHECK DATA can be used to reset the CHECK pending status.

LOAD ENFORCE CONSTRAINTS is not equivalent to CHECK DATA. LOAD ENFORCE CONSTRAINTS deletes any rows causing 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 being loaded.

When loading referential structures with ENFORCE CONSTRAINTS, parent tables should be loaded before dependent tables.

Compressing data

You can use LOAD with the REPLACE or RESUME NO options to build a *compression dictionary.* 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. To save processing costs, an initial LOAD does not go back to compress the records used to build the dictionary.

The number of records required to build a dictionary is dependent on the frequency of patterns in the data. For large data sets, the number of rows required to build the dictionary is a small percentage of the total number of rows to be compressed. For the best compression results, it is best to go ahead and build a new dictionary whenever you load the data.

However, there are circumstances in which you might want to compress data using an existing dictionary. If you are satisfied with the compression you are getting with an existing dictionary, you can keep that dictionary by using the KEEPDICTIONARY option of LOAD REPLACE or REORG. For both LOAD and REORG, this method also saves you the processing overhead of building the dictionary.

Consider using KEEPDICTIONARY if the last dictionary was built by REORG; REORG'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 2-16. REORG TABLESPACE” on page 277 and “Chapter 2-19. RUNSTATS” on page 375 for more information about using REORG to compress data and 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 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 11 on page 159.

```
LOAD DATA
  REPLACE KEEPDICTIONARY
  INTO TABLE DSN8610.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 11. Example of LOAD with the KEEPDICTIONARY option

You can also specify KEEPDICTIONARY for specific partitions of a partitioned table space. 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 DataPropagator NonRelational (DPropNR) licensed program. DPropNR runs as an MVS application and can extract data from VSAM and physical sequential access method (SAM) files as well from DL/I databases.

Using DPropNR, 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 tell which fields to extract and which conditions, if any, the source records or segments must meet.

With JCL models you edit, you can have DPropNR produce the statements for a DB2 LOAD utility job. If you have more than one DB2 system, you can name the one to receive the output. DPropNR can generate LOAD control statements in the job to relate fields in the extracted data to target columns in DB2 tables.

You can choose whether DPropNR writes the extracted data as either:

- 80-byte records included in the generated job stream
- A separate physical sequential data set, (which can be dynamically allocated by DPropNR) with a logical record length long enough to accommodate any row of the extracted data.

In the first case, the LOAD control statements generated by DPropNR include the CONTINUEIF option to describe the extracted data to DB2 LOAD.

In the second case, you can have DPropNR name the data set containing the extracted data in the SYSREC DD statement in the LOAD job. (In that case, DPropNR makes no provision for transmitting the extracted data across a network.)

Normally, you do not have to edit the job statements produced by DPropNR. However, in some cases you might have to edit; for example, if you want to load character data into a DB2 column with INTEGER data type. (DB2 LOAD does not consider CHAR and INTEGER data compatible.)

DPropNR is a versatile tool that contains more control, formatting, and output options than are described here. For more information about them, see *DataPropagator NonRelational MVS/ESA Administration Guide.*

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 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 produced by an inline copy contains ICTYPE=F, SHRLEVEL=R. The STYPE column contains an R if the image copy was produced by LOAD REPLACE LOG(YES), and an S if the image copy was produced by LOAD REPLACE LOG(NO). The data set 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 will be out of sequence and might be repeated
- If the compression dictionary is rebuilt with LOAD, the set of dictionary pages will occur twice in the data set, with the second set being the correct one.

The total number of duplicate pages will be small, with a negligible effect on the space required 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:

- Use one LOAD DATA statement when loading multiple tables in the same table space. Follow the LOAD statement with multiple INTO TABLE WHEN statements.
- Run LOAD concurrently against separate partitions of a partitioned table space.
- Preprocess the input data. For more information about preprocessing input data, see “Before running LOAD” on page 149.
- Load numeric data in its internal representation.
- Avoid data conversion, such as integer to decimal or 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 having to reorganize it after loading.
- Skip the sort phase of LOAD. The sort phase will be skipped when your input data meets all of the following conditions:
  - There is no more than one key per table.
  - All keys are of the same type (for example, all index keys, all indexed foreign keys, all nonindexed foreign keys, and so on).
  - The data being loaded is in key order.

  - The data being loaded is grouped by table and each input record must be loaded into one table only.
- If you cannot skip the sort phase because one or more of the conditions stated in the previous bullet are not met, use the SORTKEYS keyword to improve the efficiency of the sort.
- 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 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 it's an index key, sort the data in either ascending or descending order, depending on how the index was defined. If it's 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 above 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 with SORTKEYS

Use the SORTKEYS keyword to improve performance of the index key sort.

Advantages of using SORTKEYS: If you use SORTKEYS, index keys are passed to sort in memory rather than written to work files. Avoiding this I/O to the work files improves LOAD performance.

You also reduce DASD space requirements for the SYSUT1 and SORTOUT data sets, especially if you provide an estimate of the number of keys to sort.

Using the SORTKEYS option reduces the elapsed time from the start of the reload phase to the end of the build phase.

However, if the index keys are already in sorted order, or there are no indexes, SORTKEYS does not provide any advantage.

| You can reduce the elapsed time of a LOAD job for a table space or partition with
| more than one index defined by specifying the parameters to invoke a parallel
| index build. For more information, see “Building indexes in parallel for LOAD” on page 165.

Estimating the number of keys: You can specify an estimate of the number of keys for the job to sort. If the estimate is omitted or 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 steps above for each table and sum the results.

Sort data sets: If you specify the SORTKEYS keyword and omit or specify as 0 an estimate of the number of keys to be sorted, utility processing requires the sort input data set (SYSUT1) and the sort output data set (SORTOUT). See page 150 for instructions on calculating the size of those data sets.

Improving performance with LOAD or REORG PREFORMAT

When DB2's preformatting delays impact the performance or execution time consistency of high INSERT applications and the table size can be predicted for a business processing cycle, LOAD PREFORMAT or REORG PREFORMAT might be a technique to consider. This technique will only be of value if DB2's preformatting causes a measurable delay with the INSERT processing or causes inconsistent elapsed times for INSERT applications. It is recommended that a performance assessment be conducted before and after LOAD or REORG PREFORMAT is used to quantify its value in your environment.

Considerations for using PREFORMAT: PREFORMAT is a technique used to eliminate DB2 having to preformat new pages in a table space during execution time. This might eliminate execution time delays but adds the preformatting cost as setup prior to the application's execution. LOAD or REORG PREFORMAT primes a new table space and prepares it for INSERT processing. When the preformatted space is utilized and DB2 has to extend the table space, normal data set extending and preformatting occurs.

Preformatting for INSERT processing may be desirable for high INSERT tables that will receive a predictable amount of data allowing all the required space to be pre-allocated prior to the application's execution. This would be the case for a table that acts as a repository for work items coming into a system that are subsequently used to feed a backend task that processes the work items.

Preformatting of a table space containing a table used for query processing may 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 will not delete and reallocate them during utility processing. The size of the data set will not shrink back to the original data set allocation size but will either remain the same or increase in size if additional space or data is added. This has implications when LOAD or REORG PREFORMAT is used because preformatting causes all free pages between the high-used RBA (or page) to the high-allocated RBA to be preformatted. This includes secondary extents that may have been allocated.

For DB2 managed data sets, DB2 will delete and reallocate them if you specify REPLACE on the LOAD or REORG job. This will result in the data sets being re-sized to their original allocation size. They will remain that size if the data being reloaded does not fill the primary allocation and force a secondary allocation. This means the LOAD or REORG PREFORMAT option with DB2 managed data sets will at minimum cause the full primary allocation amount of a data set to be preformatted following 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 the 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 processing required to preformat all pages between the data loaded or reorganized and the high-allocated RBA. The additional LOAD or REORG time required depends on the amount of DASD space being preformatted.

Table space scans can also be elongated because empty preformatted pages will be read. It is best to 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 non-indexed queries against a preformatted table space may impact the query performance without providing a compensating improvement in the insert performance. Best results may be seen where there is a high ratio of inserts to read operations.

Considerations for running LOAD

This section describes additional points to keep in mind when running LOAD.

Be aware that running the LOAD utility on a table space does not activate triggers defined on tables in the table space.

Converting input data

The LOAD utility converts data between compatible data types.[2]

Tables 20, 21, 22, and 23 identify the allowable data conversions and the defaults used when you do not specify the input data type in a field specification of the INTO TABLE statement.

[2] The source type is used for user-defined distinct types.

Table 20. Numeric data conversion

Input Data Types	Output Data Types			
	SMALLINT	INTEGER	DECIMAL	FLOAT
SMALLINT	Default	Allowed	Allowed	Allowed
INTEGER	Allowed	Default	Allowed	Allowed
DECIMAL	Allowed	Allowed	Default	Allowed
FLOAT	Allowed	Allowed	Allowed	Default

Table 21. Character data conversion

Input Data Types	Output Data Types				
	BLOB	CHAR	CLOB	VARCHAR	LONG VARCHAR
CHAR	Allowed	Default	Allowed	Allowed	Allowed
CHAR MIXED	Allowed		Allowed		
VARCHAR	Allowed	Allowed	Allowed	Default	Default

Table 22. Time data conversion

Input Data Types	Output Data Types		
	DATE	TIME	TIMESTAMP
DATE EXTERNAL	Default		
TIME EXTERNAL		Default	
TIMESTAMP EXTERNAL	Allowed	Allowed	Default

Table 23. Graphic data conversion

Input Data Types	Output Data Types			
	DBCLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC
GRAPHIC	Allowed	Default	Allowed	Allowed
VARGRAPHIC	Allowed	Allowed	Default	Default

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 in the following cases:

- The ASCII option is specified (the input data is in ASCII) and the table space is EBCDIC.

- The EBCDIC option is specified or defaulted (the input data is in EBCDIC) and the table space is ASCII.
- 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 there is no DISCARDS processing
- To map the input record for subsequent discarding and continue (if there is DISCARDS processing)

Truncation of the decimal part of numeric data is not considered a conversion error.

Specifying input fields

When specifying input fields, consider:

- 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, or
- Specify decimal points explicitly in the input file.

Building indexes while loading data

LOAD builds all the indexes defined for any table being loaded. At the same time, it checks for duplicate values of any unique index key. If there are any duplicate values, none of the corresponding rows is loaded. Error messages identify the input records that produce duplicates; and, optionally, the records are copied to a discard data set. At the end of the job, a summary report lists all errors found.

For unique indexes, any two null values are taken 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, though 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

Use parallel index build to reduce 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 emits its first sorted record. For more information about improving index key sort performance, see “Improving performance with SORTKEYS” on page 161.

LOAD uses parallel index build if all of the following conditions are true:

- There is more than one index to be built.
- The LOAD job specifies the SORTKEYS keyword, *along with a non-zero estimate of the number of keys,* in the utility statement.
- You either allow the utility to dynamically allocate the data sets needed by SORT, or provide the necessary data sets yourself.

For a diagram of parallel index build processing, see Figure 18 on page 321.

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 SORTKEYS and SORTDEVT keywords 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: Allows you to control allocation of sort work data sets, while LOAD allocates message data sets.

1. Specify the SORTKEYS keyword in the utility statement.
2. Provide DD statements with DDNAMEs in the form SW*nn*WK*mm*.
3. Allocate UTPRINT to SYSOUT.

Method 3: Allows the most control over rebuild processing; you must specify both sort work and message data sets.

1. Specify the SORTKEYS keyword in the utility statement.
2. Provide DD statements with DDNAMEs in the form SW*nn*WK*mm*.
3. Provide DD statements with DDNAMEs in the form UTPRIN*nn*.

Data sets used: If you select Method 2 or 3 above, use the information provided here along with "Determining the number of sort subtasks" on page 167, "Allocation of sort subtasks" on page 167, and "Estimating the sort work file size" on page 167 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 DASD free space.
- To limit the number of utility subtasks used to build indexes.

The DDNAMEs SW*nn*WK*mm* define the sort work data sets used during utility processing. *nn* identifies the subtask pair, while *mm* identifies one or more data sets to be used by that subtask pair. For example:

SW01WK01 The first sort work data set used by the subtask building the first index.

SW01WK02 The second sort work data set used by the subtask building the first index.

SW02WK01 The first sort work data set used by the subtask building the second index.

SW02WK02 The second sort work data set used by the subtask building the second index.

The DDNAMEs UTPRIN*nn* define the sort work message data sets used by the utility subtask pairs. *nn* identifies the subtask pair.

Determining the number of sort subtasks: The maximum number of utility subtask pairs started for parallel index build is equal to the number of indexes 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 allocated.
- The number of subtask pairs equals the number of message data sets allocated.
- If you allocate both sort work and message data set groups, the number of subtask pairs equals the smallest number of data sets allocated.

Allocation of sort subtasks: LOAD attempts to assign one sort subtask pair for each index 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 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 24 for conceptual information about subtask pairing when the number of indexes (seven indexes) exceed the available number of subtask pairs (five subtask pairs).

Table 24. LOAD subtask pairing for a relational table space

Subtask Pair	Index Assigned
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 will need to know the size and number of keys present in all of the indexes being processed by the subtask in order to calculate each sort work file size. After you've

determined which indexes are assigned to which subtask pairs, use the following formula to calculate the space required:

2 × (longest index key + 14) × (number of keys extracted)

longest key The length of the longest key that will be processed by the subtask. For the first subtask pair for LOAD, compare the length of the longest key and the longest foreign key, and use the largest value.

number of keys The number of keys from all indexes to be sorted that will be processed by the subtask.

Leaving free space

When loading into a nonsegmented table space, LOAD leaves one free page after reaching the FREEPAGE limit, regardless of whether the records loaded 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. (Those values can be set by 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 or REBUILD pending status

You cannot load records 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 being loaded is in the REBUILD pending status. for information about resetting the RECOVER pending status. See “Resetting the REBUILD pending status” on page 218 for information about resetting the REBUILD pending status.

If you are replacing a partition, these restrictions are relaxed; the partition being replaced can be in the RECOVER pending status, and its corresponding index partition can be in the REBUILD pending status. However, all nonpartitioning indexes must *not* be in the page set REBUILD pending status. See Appendix C, “Resetting an advisory or restrictive status” on page 527 for more information about resetting a restrictive status.

There is one RECOVER pending restrictive status:

RECP The table space or partition is in the RECOVER pending status. If a single logical partition is in RECP, the partition is treated as RECP for SQL access. A single logical partition in RECP does not restrict utility access to other logical partitions not in RECP. RECP is reset by recovering only the single logical partition.

There are three REBUILD pending restrictive states:

RBDP REBUILD pending status (RBDP) is set on a physical or logical index partition. The individual physical or logical partition is inaccessible and must be rebuilt using the REBUILD INDEX utility, or recovered using the RECOVER utility.

PSRBD Page set REBUILD pending (PSRBD) is set for nonpartitioning indexes. The entire index space is inaccessible and must be rebuilt using the REBUILD utility, or recovered using the RECOVER utility.

RBDP* RBDP* (REBUILD pending star) status is set only on logical partitions of nonpartitioning indexes. The entire index is inaccessible, but is made available again when the affected partitions are rebuilt using the REBUILD INDEX utility, or recovered using the RECOVER utility.

See Table 93 on page 531 for information about resetting the RECOVER pending status, and Table 92 on page 530 for information about resetting the REBUILD pending status.

Using exit procedures

Any field procedure associated with a column of a table 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 columns defined as ROWID

Columns defined as ROWID can be designated as input fields using 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 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: Columns defined as ROWID GENERATED BY DEFAULT can be set by the LOAD utility from input data. 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 will occur. If such an error occurs, the load will fail. In this case, you need to discard the duplicate value and retry the load 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 will be loaded with a value 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. When the input record is greater than 32KB, you might have to load the LOB data separately. See sample job DSN8DLPL in SDSNSAMP for an example.

Using LOAD LOG on a LOB table space

A LOB table space that was defined with LOG YES or LOG NO will affect logging while loading a LOB column. Table 25 shows the logging output and LOB table space effect, if any.

Table 25. 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
LOG NO	LOG NO	Nothing	COPY Pending

Inline statistics collection 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, LOAD utility processing collects inline statistics prior to discarding rows as a result of 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.

Inline COPY for a base table space

If you take an inline image copy of a table with LOB columns, DB2 makes a copy of the base table space, but does not copy the LOB table spaces.

Terminating or restarting LOAD

For instructions on restarting a utility job, see “Restarting an online utility” on page 48.

Terminating LOAD: If you terminate LOAD 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 using the TERM UTILITY command during the sort or build phases, then the indexes not yet built remain in the REBUILD pending status.

If you use the SORTKEYS option and the LOAD job terminates during the RELOAD, SORT, BUILD, or SORTBLD phases, then both RESTART and RESTART(PHASE) 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 will result in message DSNU257I.

Table 26. LOAD phases and pending statuses

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. Resets COPY pending at end of phase if an inline copy is produced unless SORTKEYS is also specified.
Build	Resets REBUILD pending status for nonunique indexes. Resets COPY pending status at end of phase if an inline copy is produced and SORTKEYS is also specified.
Indexval	Resets REBUILD pending status for unique indexes.
Enforce	Resets CHECK pending status for table space.

Restarting LOAD: Table 27 on page 172 provides information about restarting LOAD, depending on the phase LOAD was in when the job stopped.

- If you restart LOAD during the UTILINIT phase, it re-executes from the beginning of the phase.
- If LOAD abends or system failure occurs while it is in the UTILTERM phase, you must restart with RESTART(PHASE).
- If you restart a LOAD job for a table with LOB columns that specified the RESUME YES option, you must use RESTART CURRENT.
- If you use RESTART PHASE to restart a LOAD job which specified RESUME NO, the LOB table spaces and indexes on auxiliary tables will be reset.
- If you restart a LOAD job which uses the STATISTICS keyword, inline statistics collection will not occur. To update catalog statistics, run the RUNSTATS utility after the restarted LOAD job completes.

In this table, the TYPE column distinguishes between the effects of specifying RESTART or RESTART(PHASE).

Table 27. LOAD restart information

Phase	Type	Data Sets Required	Notes
RELOAD	CURRENT	SYSREC and SYSUT1	
		SYSMAP and SYSERR	2,3
	PHASE	SYSREC	7
SORT	CURRENT	SYSUT1	1
	PHASE	SYSUT1	
BUILD	CURRENT	SORTOUT	1,5
	PHASE	SORTOUT	5
SORTBLD	CURRENT	SYSUT1 and SORTOUT	5,9
	PHASE	SYSUT1 and SORTOUT	5,9
INDEXVAL	CURRENT	SYSERR or SYSUT1	3
	PHASE	SYSERR or SYSUT1	3
ENFORCE	CURRENT	SORTOUT and SYSUT1	4
	PHASE	SORTOUT and SYSUT1	4
DISCARD	CURRENT	SYSMAP and SYSERR	4
		SORTOUT and SYSUT1	8
	PHASE	SYSMAP and SYSERR	4
		SORTOUT and SYSUT1	8
REPORT	CURRENT	SYSERR or SORTOUT	4
		SYSMAP and SYSERR	6
	PHASE	SYSERR or SORTOUT	4
		SYSMAP and SYSERR	6

Note:

1. 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.
2. SYSMAP and SYSERR data sets may not be required for all load jobs. See “Chapter 2-9. LOAD” on page 121 for exact requirements.
3. 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.
4. 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 used and whether any input data sets have been rewritten.
5. LOAD RESUME YES cannot be restarted in the BUILD or SORTBLD phase.
6. If report is required and this is a load without discard processing, SYSMAP is required to complete the report phase.
7. You must not restart during RELOAD phase if you specified SYSREC DD *. This 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 consists of multiple, concatenated data sets.
8. The SYSUT1 data set is required if the target table space is segmented or partitioned.
9. If you specified SORTKEYS, then use RESTART or RESTART(PHASE) to restart at 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 49 for guidance in restarting LOAD from the last commit point after receiving an out of space condition.

Concurrency and compatibility

Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions of the same table space or index space are compatible.

For nonpartitioning indexes, LOAD PART:

- Drains only the logical partition
- Does not set the page set REBUILD pending status (PSRBD)
- Does not respect PCTFREE or FREEPAGE attributes when inserting keys

Claims and drains

Table 28 shows which claim classes LOAD drains and the restrictive states the utility sets.

Table 28. Claim classes of LOAD operations. Use of claims and drains; restrictive states set on the target object.

Target	LOAD	LOAD PART
Table space, index, or physical partition of a table space or index	DA/UTUT	DA/UTUT
Nonpartitioning index	DA/UTUT	DR
Index logical partition		DA/UTUT
Primary index (with ENFORCE option only)	DW/UTRO	DW/UTRO
RI dependents	CHKP (NO)	CHKP (NO)

Legend:

- CHKP (NO): Concurrently running applications will not see CHECK PENDING after commit
- 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
- Blank: Object is not affected by this utility

Compatibility

The following 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.

- DIAGNOSE
- REPORT

- STOSPACE

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 have used 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 specified RESUME NO or REPLACE, indicating that this is the first load into the table space, we recommend that you take two or more full image copies to enable recovery. Alternatively, we recommend that you take primary and backup
| inline copies when you do a LOAD REPLACE; full table space or partition image
| copies taken after the LOAD completes are not necessary. However, you might
| need to take images copies of indexes.

| Use either the STATISTICS option to collect inline statistics, or the RUNSTATS utility so that the DB2 catalog statistics take into account the newly loaded data, and DB2 can select SQL paths with accurate information. Following this, rebind any application plans that depend on the loaded tables to update the path selection of any embedded SQL statements.

Resetting the 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 (though 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 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 will take longer than if you had made an image copy.

A table space in COPY pending status can be read without restriction; however, it cannot be updated.

Resetting the REBUILD pending status

LOAD places all the index spaces for a table space in the REBUILD pending status if you end the job (using -TERM UTILITY) before it completes the INDEXVAL phase. It places the table space itself in RECOVER pending if you end the job before it completes the RELOAD phase.

Resetting the RECOVER pending status depends on when the utility terminated:

- If the data is intact (running the -DISPLAY DATABASE command shows indexes are in REBUILD pending status but the table space is not in RECOVER pending status), you can recover the indexes using RECOVER INDEX, if you have a full image copy of the affected indexes. If you do not have an image copy available, you must rebuild the entire index using the REBUILD INDEX utility. However, for partitioning indexes and for nonpartitioning indexes in REBUILD pending (RBDP), you can use the PART option of REBUILD INDEX to rebuild separate partitions of the index.
- If the data is not intact (running the -DISPLAY DATABASE command shows the table space is in RECOVER pending status), you can either load the table again or recover it to a prior point of consistency. 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. That utility locates invalid data and, optionally, removes it. If it removes the invalid data, the data remaining satisfies all check and referential constraints and the CHECK pending restriction is lifted.

Though CHECK DATA is usually preferred, the CHECK pending status can also be reset by any of the following operations:

- Dropping tables that contain invalid rows
- Replacing the data in the table space, using LOAD REPLACE and enforcing check and referential constraints
- Recovering all members of the table space set to a prior quiesce point
- REPAIR SET with NOCHECKPEND

In the next sections, we illustrate the use of CHECK DATA after two likely LOAD jobs.

Running CHECK DATA after LOAD REPLACE: Suppose you choose to replace the contents of the project table 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 containing the project activity table to reset the status. First, give particular care to the options described below. Then, when you run the utility, make sure that all table spaces are available

that contain either parent tables or dependent tables of any table in the table spaces 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.* At this point, we assume that you already have an exception table available for every table subject to referential or table check constraints. (For instructions on creating them, see page 60.)

If you use DELETE YES, you must name an exception table for every descendent of every table in every table space being checked. Deletes caused by CHECK DATA are not subject to any of the SQL delete rules; they cascade without restraint to the farthest descendent.

If table Y is the exception table for table X, name it with this clause in the CHECK DATA statement:

```
FOR EXCEPTION IN X USE Y
```

Error and sort data sets

The options ERRDDN, WORKDDN, SORTDEVT, and SORTNUM function 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/Merge or let DB2 allocate them dynamically.

The following example runs CHECK DATA against the table space containing the project activity table. It assumes the existence of exception tables named DSN8610.EPROJACT and DSN8610.EEPA.

```
CHECK DATA TABLESPACE DSN8D61A.PROJACT
   DELETE YES
   FOR EXCEPTION IN DSN8610.PROJACT USE DSN8610.EPROJACT
                 IN DSN8610.EMPPROJACT USE DSN8610.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 like these:

```
//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)
//SORTWK01 DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTWK02 DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTWK03 DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTWK04 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, 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 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, it is *not* in CHECK pending status.
- Because you did not enforce constraints on the project activity table, it *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 so might not violate the referential integrity of its dependent. But if you delete records from PROJACT when checking, you still need an exception table for EMPPROJACT.

Hence you want to check the data in the project activity table.

SCOPE PENDING

DB2 records the identifier of the first record 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 records that might be in error.

The following example runs CHECK DATA against the table space containing the project activity table after LOAD RESUME:

```
CHECK DATA TABLESPACE DSN8D61A.PROJACT
   SCOPE PENDING
   DELETE YES
   FOR EXCEPTION IN DSN8610.PROJACT USE DSN8610.EPROJACT
                 IN DSN8610.EMPPROJACT USE DSN8610.EEPA
   SORTDEVT SYSDA
   SORTNUM 4
```

As before, the JCL data set for the job needs DD statements to define data sets for the error and sort data sets.

Collecting inline statistics while loading a table

If you do not specify LOAD RESUME, use the STATISTICS keyword to gather inline statistics about space use and row clustering to update the DB2 catalog. The data is used to select access paths when executing SQL statements. This procedure eliminates the need to run RUNSTATS after loading a table space. However, it you perform a LOAD PART operation, you should run RUNSTATS INDEX on the nonpartitioning indexes to update that data.

Running CHECK INDEX after loading a table having 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 the output of RUNSTATS), 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, the SYSCOPY record is inserted at the beginning of the RELOAD phase if LOG YES was 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.

Sample control statements

Example 1: LOAD JCL with RESUME YES and ENFORCE NO. This example shows the JCL for loading a table with the RESUME YES and ENFORCE NO options. This job will place the table in the CHECK pending status.

```
| //STEP1    EXEC DSNUPROC,UID='IUIQU2UB.LOAD',
| //         UTPROC='',
| //         SYSTEM='V61A'
| //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 DSN8610.ACT
          (ACTNO    POSITION( 1) INTEGER EXTERNAL(3),
           ACTKWD   POSITION( 5) CHAR(6),
           ACTDESC  POSITION(13) VARCHAR)
     ENFORCE NO
//*
```

Example 2: Control statement with RESUME YES option. Figure 12 on page 179 shows a LOAD utility statement. It loads the records from the data set named by the SYSREC DD statement for the utility job into the department table. The RESUME YES clause specifies that the table space need not be empty; new records are added at the end.

```
LOAD DATA
  RESUME YES
  INTO TABLE DSN8610.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 12. Example of a LOAD utility statement

This example uses the POSITION clause to specify where a field is in the input record. With the statement above, LOAD accepts the input shown in Figure 13 on page 179 and interprets it as follows:

- The first three bytes of each record are loaded into the DEPTNO column of the table.
- The next 36 bytes are loaded into the DEPTNAME column, including trailing blanks.

  If we had chosen to define this input column as VARCHAR(36), the input data would have had to contain a 2-byte binary length field preceding the data.
- The next three fields are loaded into columns defined as CHAR(6), CHAR(3), and CHAR(16).

```
A00SPIFFY COMPUTER SERVICE DIV.        000010A00USIBMSTODB21
B01PLANNING                            000020A00USIBMSTODB21
C01INFORMATION CENTER                  000030A00USIBMSTODB21
D01DEVELOPMENT CENTER                        A00USIBMSTODB21
```

Figure 13. Records in an input data set for LOAD

Table 29 shows how the same records appear if you then execute the statement SELECT * FROM DSN8610.DEPT under SPUFI.

Table 29. Data loaded to 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 3: Load data into a table. Load data from the data set specified by the EMPLDS DD statement into the EMP table.

```
LOAD DATA INDDN EMPLDS
  INTO TABLE DSN8610.EMP
```

Example 4: Load data into two tables. Load data from the data set specified by the EMPLDS DD statement into the DSN8610.EMP and SMITH.EMPEMPL tables.

```
LOAD DATA INDDN EMPLDS
  INTO TABLE DSN8610.EMP
  INTO TABLE SMITH.EMPEMPL
```

Example 5: Load selected records into a table. Load data from the data set specified by the EMPLDS DD statement into the EMP table. Load only from source input records that begin with LKA.

```
LOAD DATA INDDN EMPLDS
  INTO TABLE DSN8610.EMP
  WHEN (1:3)='LKA'
```

Example 6: Load selected records into a non-empty table space. The data from the sequential data set identified by the SYSREC DD statement is selectively loaded into the DSN8610.DEPT table whenever positions 1 through 3 contain the value LKA. The table space need not be empty for loading to proceed.

For each source record that has LKA in its first three positions:

- 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 VARCHAR column
- The characters in positions 36 through 41 are loaded into the MGRNO column
- Characters in positions 42 through 44 are loaded into the ADMRDEPT column.

```
LOAD DATA
  RESUME YES
  INTO TABLE DSN8610.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)
```

Example 7: Load selected records into an empty table space. Data from the sequential data set identified by the SYSRECPJ DD statement is selectively loaded into the DSN8610.PROJ table. The table space containing the DSN8610.PROJ table is currently empty, because the RESUME YES option was not specified.

For each source input record, data is loaded into the specified columns (that is, PROJNO, PROJNAME, DEPTNO ..., and so on) to form a table row. Any other columns in a DSN8610.PROJ row are set to NULL.

Starting positions of the fields in the sequential data set are defined by the field specification POSITION options. The ending position of the fields in the sequential data set are implicitly defined either by the length specification of the data type options (CHAR length) or by the length specification of the external numeric data type (LENGTH).

The numeric data 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 are assumed to be represented by eight digits and two separator characters, as in the USA format (for example, 11/15/1987). The length of the date fields is given as 10 explicitly, though in many cases it defaults to the same value.

```
LOAD DATA INDDN(SYSRECPJ)
  INTO TABLE DSN8610.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))
```

Example 8: Load data selectively using the CONTINUEIF option. Data from the sequential data set specified by the SYSRECOV DD statement is assembled and selectively loaded into the DSN8610.TOPTVAL table. The table space that contains DSN8610.TOPTVAL is currently empty because the RESUME YES option is not specified.

Fields destined for columns in the same table row can span more than one source record. Source records having fields containing columns that belong to the same row as the next source record all have an X in column 72 (that is, `CONTINUEIF(72:72)='X'`).

For each assembled source record, fields are loaded into the DSN8610.TOPTVAL table columns (that is, MAJSYS, ACTION, OBJECT ..., DSPINDEX) to form a table row. Any columns not mentioned are set to NULL.

The starting positions of the fields in the assembled source record input are given in the POSITION option. Starting positions are numbered from the first column of the internally assembled input record, not from the start of the source records in the sequential data set. The ending positions are defined by the character string lengths given with the input data type.

No conversions are required to load the source character strings into their designated columns, which are also defined to be fixed character strings. However, because columns INFOTXT, HELPTXT, and PFKTXT are defined as 79 characters in length and the strings 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 DSN8610.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))
```

Example 9: Load data with referential constraints. Data from the sequential data set identified by the SYSREC DD statement is loaded into the DSN8610.PROJ. table. Referential constraints are enforced on data added. Output consists of a

summary report of violations of referential constraints, and all records causing these violations are placed in the SYSDISC discard data set.

```
LOAD DATA INDDN(SYSREC) CONTINUEIF(72:72)='X'
 RESUME YES
 ENFORCE CONSTRAINTS
 INTO TABLE DSN8610.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='      '))
```

Example 10: Load data using SORTKEYS. Use the SORTKEYS keyword to improve performance of the index key sort as shown in the following example. Assume there are 22,000 rows to load into the DSN8610.DEPT table. This table has 3 indexes.

The following job specifies an estimate of 66,000 keys to sort with the SORTKEYS keyword, using the calculation described in “Improving performance with SORTKEYS” on page 161:

```
(3 + 0) * 22,000 = 66,000
```

This example specifies dynamic allocation of the required data sets by DFSORT™, using the SORTDEVT and SORTNUM keywords. If sufficient virtual storage resources are available, one utility subtask pair will be 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.

LOAD statement:

```
//SAMPJOB  JOB  ...
//STEP1    EXEC DSNUPROC,UID='SAMPJOB.LOAD',UTPROC='',SYSTEM='V61A'
//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 DSN8610.DEPT
/*
```

Example 11: LOAD with inline copy. Use the REPLACE option with COPYDDN and RECOVERYDDN to create copies during LOAD.

```
LOAD DATA REPLACE INDDN INPUT
 SORTKEYS 66000
 COPYDDN SYSCOPY RECOVERYDDN REMCOPY
 CONTINUEIF(79:80)='++'
 INTO TABLE DSN8610.DEPT
```

Example 12: Load ASCII input data. Use the ASCII option to load ASCII input data into a table named MYASCIIT that was created with the CCSID ASCII clause.

```
LOAD REPLACE LOG NO ASCII INTO TABLE MYASCIIT
 (NAME POSITION(1) CHAR(40),
  ADDRESS POSITON(41) CHAR(40),
  ZIP POSITION(81) DECIMAL EXTERNAL(9),
  DEPARTMENT POSITION(90) CHAR(3),
  TITLE POSITION(93) GRAPHIC(20))
```

The CCSID keyword is not specified in this example; therefore, the CCSIDs of the ASCII input data are assumed to be the ASCII CCSIDs specified at installation. Conversions are done only if the CCSIDs of the target table differ from the ASCII CCSIDs specified at installation.

Example 13: Load data using statistics collection. Use the STATISTICS keyword to gather catalog statistics for the table space. This eliminates the need to run the RUNSTATS utility after completing the load operation. Specify REUSE so that all partitions are logically reset rather than deleted and redefined.

```
LOAD DATA
     INDDN SYSREC
     REPLACE
     STATISTICS TABLE(ALL)
     INDEX(ALL)
     REPORT YES UPDATE ALL
     REUSE
     CONTINUEIF(79:80)='++'
     INTO TABLE
     DSN8610.DEPT
```

Example 14: Load data for a partitioned table space using statistics collection. Load data for a specified partition, using the STATISTICS keyword to gather catalog statistics of the partitioned table space.

```
LOAD STATISTICS
  INTO TABLE DSN8610.DEPT PART 1 REPLACE
```

Chapter 2-10. MERGECOPY

The MERGECOPY online utility merges image copies produced by the COPY utility or inline copies produced by the LOAD or REORG utilities. 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.

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 control statement” on page 186. For detailed guidance on running this utility, see “Instructions for running MERGECOPY” on page 188.

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, the privilege set of the process must include one of the following:

- 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.

Execution phases of MERGECOPY: One of the following phases can be identified if the job terminates.

The phases for MERGECOPY are:

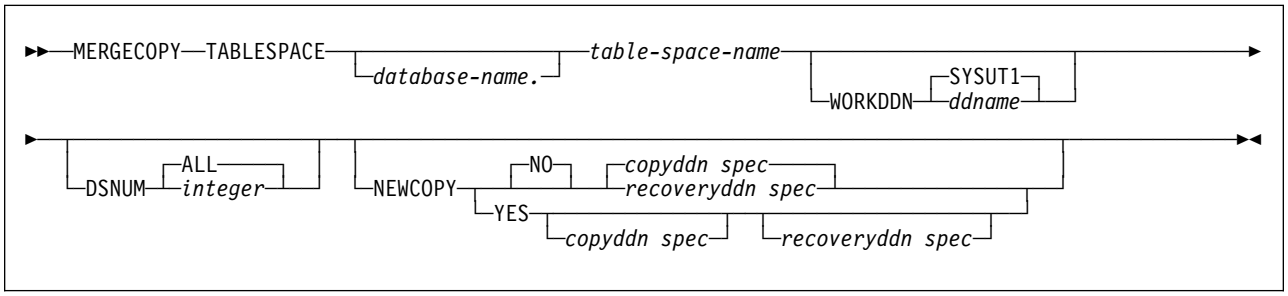
Phase	Description
UTILINIT	Initialization
MERGECOP	Merge incremental copies
UTILTERM	Cleanup.

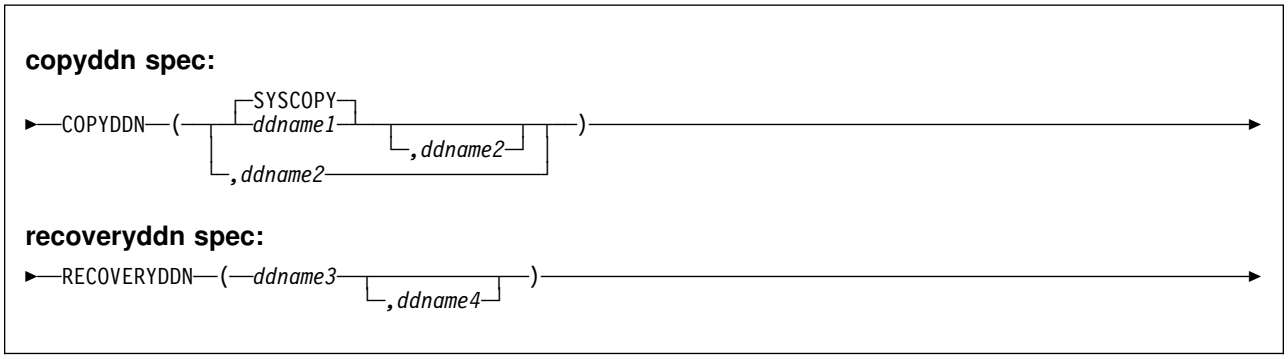
Syntax and options of the control statement

The utility control statement defines the function 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

For guidance in interpreting syntax diagrams, see "How to read the syntax diagrams" on page 4.





Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see "Control statement coding rules" on page 27.

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

Specifies the table space (and, optionally, the database it belongs to) that is to be copied.

database-name The name of the database 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.

The following are optional.

WORKDDN *ddname*
Specifies a DD statement for a temporary data set, to be used for intermediate merged output.

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, it is possible that only some of the image copy data sets will be 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.

DSNUM Identifies a partition or data set, within the table space, that is to be merged; or it merges the entire table space.

ALL Merges the entire table space. The **default** is **ALL**.

integer Is the number of a partition or data set to be merged. The maximum is 254.

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 this format:

catname.DSNDB*x*.*dbname*.*tsname*.I0001.A*nnn*

where *nnn* is the data set integer.

If image copies were taken by data set (rather than by table space), then MERGECOPY must use the copies by data set.

NEWCOPY Tells whether incremental image copies are to be merged with the full image copy or not.

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.

The **default** is **COPYDDN**(**SYSCOPY**), where SYSCOPY identifies the primary data set.

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.

There is no default for RECOVERYDDN.

Instructions for running MERGECOPY

To run MERGECOPY, you must:

1. Prepare the necessary data sets, as described in "Data sets used by MERGECOPY."
2. Create JCL statements, by using one of the methods described in "Chapter 2-1. Invoking DB2 online utilities" on page 27. (For examples of JCL for MERGECOPY, see "Sample control statements" on page 193.)
3. Prepare a utility control statement, specifying the options for the tasks you want to perform, as described in "Instructions for specific tasks" on page 189. (For a complete description of the syntax and options for MERGECOPY, see "Syntax and options of the control statement" on page 186.)
4. Check the compatibility table in "Concurrency and compatibility" on page 192 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 192.
6. Run MERGECOPY.

See "Chapter 2-1. Invoking DB2 online utilities" on page 27 for an explanation of ways to execute DB2 utilities.

Data sets used by MERGECOPY

Table 30 describes the data sets used by MERGECOPY. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 30 (Page 1 of 2). Data sets used by MERGECOPY

Data Set	Description	Required?
SYSIN	Input data set containing the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
Image copy data set	The image copy data set defines the resulting image copy. Its DD names is specified with the COPYDDN parameter of the MERGECOPY statement. The default DD name is SYSCOPY.	Yes
Work data set	This is a temporary data set that is used for intermediate merged output. Its DD name is specified through the WORKDDN parameter of the MERGECOPY statement. The default DD name is SYSUT1.	Yes

Table 30 (Page 2 of 2). Data sets used by MERGECOPY

Data Set	Description	Required?
Input data sets	These are image copy data sets which may be pre-allocated by the user. The DD names are defined by the user.	No

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

Table space
Object to be copied. It is named in the MERGECOPY control statement and is accessed through the DB2 catalog.

Data sets: The input data sets for the merge operation are dynamically allocated. To merge incremental copies, a work data set (WORKDDN) and up to two new copy data sets (COPYDDN) are allocated in the JCL for the utility job. These can be allocated to tape or DASD. If these allocations are made to tape, an additional tape drive is required for each of those data sets.

The COPYDDN option of MERGECOPY allows you to specify the ddnames for the output data sets. The option has the format `COPYDDN (ddname1,ddname2)`, where *ddname1* is the ddname for the primary output data set in the system currently running DB2 and *ddname2* is the ddname for the backup output data set in the system currently running DB2. The default for *ddname1* is SYSCOPY.

The RECOVERYDDN option of MERGECOPY allows you to specify the output image copy data sets at the recovery site. The option has the format `RECOVERYDDN (ddname3, ddname4)`, where *ddname3* is the ddname for the primary output image copy data set at the recovery site and *ddname4* is the ddname 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 being merged. Use the same DCB attributes as used for the image copy data sets.

Creating the control statement

See “Syntax and options of the control statement” on page 186 for MERGECOPY syntax and option descriptions. See “Sample control statements” on page 193 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

The NEWCOPY parameter decides if the new copy created by MERGECOPY is an incremental image copy or a full image copy. In general, it is recommended to create a new full image copy. The reasons for this recommendation are:

- A new full image copy creates a new recoverable point.
- The additional time it takes to create a new full image copy does not have any adverse affect on the access to the table space. The only concurrency implication is the access to SYSIBM.SYSCOPY.
- The range of log records to be applied by RECOVER is the same for both the new full image copy and the merged incremental image copy.
- Assuming the copies are on tape, only one tape drive will be 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 SYSIBM.SYSCOPY records of the incremental image copies that were merged with an entry for the new incremental image copy
- Deletes all SYSIBM.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, the use of MERGECOPY to create full image copies should be considered.

Use MERGECOPY NEWCOPY YES immediately after each incremental image copy:

- Dates will become a valid criterion for image copy data set and archive log deletion.
- A minimum number of tape drives will be allocated for MERGECOPY and RECOVER execution.

Merging inline copies

If you merge an inline 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 160 for additional information about inline copies.

Using MERGECOPY with individual data sets

MERGECOPY can be used on copies of an entire table space or individual data sets or 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 produces the following messages:

```
DSNU460I DSNUBCLO - IMAGE COPIES INCONSISTENT.
                    MERGECOPY REQUEST REJECTED
DSNU010I DSNUGBAC - UTILITY EXECUTION COMPLETE,
                    HIGHEST RETURN CODE=4
```

With the option NEWCOPY YES, 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 you want to merge reside tape, refer to “Retaining tape mounts” on page 249 for general information about specifying the appropriate parameters on the DD cards.

Deciding between 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, but 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. (And you do that if you run MODIFY RECOVERY specifying the date when MERGECOPY was run as the value of DATE.)

To delete all log information that is included in a copy made by MERGECOPY:

1. Find the record of that copy in the catalog table SYSIBM.SYSCOPY. You can find it by selecting on 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 made before that date.

RECOVER uses the LOG RBA of image copies to determine the starting point in the log needed for recovery. Normally, there is a direct correspondence between a timestamp and a LOG RBA. Because of this, and because MODIFY uses dates to cleanup recovery history, you may decide to use dates to delete old archive log tapes. This may cause a problem if MERGECOPY is used. MERGECOPY inserts the LOG RBA of the last incremental image copy into the SYSIBM.SYSCOPY row created for the new image copy. The date recorded in ICDATE column of SYSIBM.SYSCOPY row is the date MERGECOPY was executed.

Creating image copies in a JES3 environment

Ensure that there are sufficient units available to mount the required image copies. In a JES3 environment, if the number of image copies to be restored exceeds the number of available online and offline units, and the MERGECOPY job successfully allocates all available units, the job will then wait 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 restarting a utility job, see “Restarting an online utility” on page 48.

You can restart a MERGECOPY utility job at the beginning of any of the phases listed below:

UTILINIT	Initialization and setup
MERGECOPY	Merge
UTILTERM	Cleanup.

Restarting MERGECOPY after an out of space condition: See “Restarting after the output data set is full” on page 49 for guidance in restarting MERGECOPY from the last commit point after receiving an out of space condition.

Concurrency and compatibility

Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions of the same table space or index space are compatible.

Table 31 shows the restrictive state the utility sets on the target object.

Table 31. Claim classes of MERGECOPY operations. Use of claims and drains; restrictive states set on the target object.

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**:

- COPY TABLESPACE
- LOAD
- MERGECOPY
- MODIFY
- RECOVER
- REORG TABLESPACE

The target object can be a table space or partition.

Sample control statements

Example 1: Creating a merged incremental copy. Create a merged incremental image copy of table space DSN8S61C.

```
//STEP1    EXEC DSNUPROC,UID='IUJMU107.MERGE1',
//          UTPROC='',SYSTEM='V61A'
//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 DSN8D61P.DSN8S61C
  COPYDDN      (COPY1,COPY2)
  NEWCOPY  NO
```

Example 2: Creating a merged full image copy. Create a merged full image copy of table space DSN8S61C.

```
//STEP1    EXEC DSNUPROC,UID='IUJMU107.MERGE2',
//          UTPROC='',SYSTEM='V61A'
//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 DSN8D61P.DSN8S61C
  COPYDDN      (COPY1,COPY2)
  NEWCOPY  YES
```

Chapter 2-11. MODIFY

The MODIFY online 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. 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 clear 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 utility automatically removes the SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX recovery records that meet the AGE and DATE criteria for all indexes over the table space that were defined with the COPY YES attribute.

For a diagram of MODIFY syntax and a description of available options, see “Syntax and options of the control statement” on page 196. For detailed guidance on running this utility, see “Instructions for running MODIFY” on page 197.

Output: MODIFY deletes image copy rows from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX.

For each full and incremental SYSCOPY record deleted from SYSCOPY, the utility returns a message giving the name of the copy data set.

For information on deleting SYSLGRNX rows, see “Deleting SYSLGRNX rows” on page 199.

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.

Authorization required: To execute this utility, the privilege set of the process must include one of the following:

- IMAGCOPY privilege for the database to run MODIFY
- DBADM, DBCTRL, or DBMAINT authority for the database
- SYSCTRL or SYSADM authority.

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

There are no SYSCOPY or SYSLGRNX records for DSNDB06.SYSCOPY, DSNDB01.SYSUTIL, or DSNDB01.DBD01. You can run MODIFY 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: One of the following phases can be identified if the job terminates.

The phases for MODIFY are:

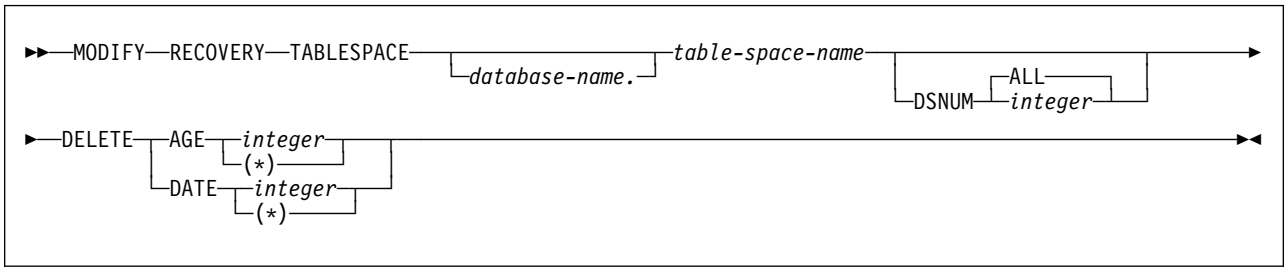
UTILINIT	Initialization and setup
MODIFY	Deleting records
UTILTERM	Cleanup.

Syntax and options of the control statement

The utility control statement defines the function 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

For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.



Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see “Control statement coding rules” on page 27.

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 254.

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, then 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 this format:

catname.DSNDB*x*.*dbname*.*tsname*.I0001.A*nnn*

where: *nnn* is the data set integer.

If you specify DSNUM, MODIFY does not delete any SYSCOPY records for the partition 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.

DELETE Indicates that records are to be deleted. See the DSNUM description above for restrictions on deleting partition statistics. on deleting partition statistics.

AGE *integer* Deletes all SYSCOPY records older than a specified number of days.

integer is the number of days, and can range from 0 to 32767. Records 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 written before a specified date.

integer may use either an 8 or 6 character format. You must specify a year (*yyyy* or *yy*), month (*mm*), and day (*dd*) in the form *yyyymmdd* or *yymmdd*. DB2 processing queries the system clock and converts 6-character dates to the most recent, previous 8-character equivalent.

(*) deletes all records, regardless of the date on which they were written.

Instructions for running MODIFY

To run MODIFY you must:

1. Read “Before running MODIFY” on page 198 in this chapter.
2. Prepare the necessary data sets, as described in “Data sets used by MODIFY” on page 198.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27. (For examples of JCL for MODIFY, see “Sample control statements” on page 200.)
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 198. (For a complete description of the syntax and options for MODIFY, see “Syntax diagram” on page 196.)
5. Check the compatibility table in “Concurrency and compatibility” on page 200 if you want to run other jobs concurrently on the same target objects.

6. Plan for restart if the MODIFY job doesn't complete, as described in "Terminating or restarting MODIFY" on page 199.
7. Run MODIFY.

See "Chapter 2-1. Invoking DB2 online utilities" on page 27 for an explanation of ways to execute DB2 utilities.

Before running MODIFY

Printing SYSCOPY records with REPORT RECOVERY: If you use MODIFY RECOVER to delete SYSCOPY records, we recommend that you first 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 RECOVER on a table space that is in RECOVER pending status. See "Chapter 2-14. RECOVER" on page 225 for information about resetting the RECOVER pending status.

Data sets used by MODIFY

Table 32 describes the data sets used by MODIFY. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 32. Data sets used by MODIFY

Data Set	Description	Required?
SYSIN	Input data set containing 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 card in the JCL:

Table space
Object for which records are to be deleted. It is named in the MODIFY control statement and is accessed through the DB2 catalog.

Creating the control statement

See "Syntax diagram" on page 196 for MODIFY syntax and option descriptions. See "Sample control statements" on page 200 for examples of MODIFY 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 rows" on page 199
"Deleting all image copy entries" on page 199
"Deleting recovery rows for indexes" on page 199
"Reclaiming space in the DBD" on page 199
"Improving REORG performance after adding a column" on page 199

Deleting SYSLGRNX rows

If you take image copies at the data set level or partition level only, then you can use MODIFY with the DSNUM option to delete SYSLGRNX rows for the data set or partition. For partitioned table spaces, MODIFY deletes partition-level SYSLGRNX records only if there are no image copies of the entire table space. Otherwise, if full copies exist for the entire table space and you specify DSNUM, MODIFY returns a message saying that no records were deleted.

If you omit DSNUM or specify DSNUM ALL, then MODIFY deletes all SYSLGRNX records pertaining to the entire table space and individual data sets and partitions.

Deleting all image copy entries

MODIFY allows you to delete all image copy entries for a table space or data set. In this case MODIFY:

- Issues message DSNU572I.
- Sets the COPY pending restriction.
- Gives return code 4.

Deleting recovery rows for indexes

When you perform MODIFY RECOVERY on a table space, utility processing deletes SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX rows that meet the AGE and DATE criteria for related indexes that were defined with COPY YES.

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 with the DELETE option to delete all previous image copies.

Improving REORG performance after adding a column

After you add a column to a table space, the next REORG of the table space materializes default values for the added column by decompressing all rows of the table space during the UNLOAD phase and then compressing them again 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 with the DELETE option to delete all previous image copies. MODIFY changes the alter added column status only if there are SYSCOPY rows to delete.

Terminating or restarting MODIFY

MODIFY can be terminated in any phase without any integrity exposure.

You are permitted to restart the MODIFY utility, but it starts from the beginning again.

For more guidance in restarting online utilities, see “Restarting an online utility” on page 48.

Concurrency and compatibility

Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions of the same table space or index space are compatible.

Table 33 shows the restrictive state the utility sets on the target object.

Table 33. Claim classes of MODIFY operations. Use of claims and drains; restrictive states set on the target object.

Target	MODIFY
Table space or partition	UTRW

Legend:

UTRW - Utility restrictive state - Read/Write access allowed.

MODIFY can run concurrently on the same target object with any utility **except the following**:

- COPY TABLESPACE
- LOAD
- MERGECOPY
- MODIFY
- RECOVER TABLESPACE
- REORG TABLESPACE

The target object can be a table space or partition.

Sample control statements

Example 1: Delete SYSCOPY records by age. For the table space containing the employee table, delete all SYSCOPY records older than 90 days.

```
//STEP1    EXEC DSNUPROC,UID='IUIQU2UD.MODRCV1',
//            UTPROC='',SYSTEM='V61A'
//SYSIN    DD *
MODIFY RECOVERY TABLESPACE DSN8D61A.DSN8S61E DELETE AGE(90)
/*
```

Example 2: Delete SYSCOPY records by date. For the table space containing the department table, delete all SYSCOPY records written before 10 September 1998.

```
MODIFY RECOVERY TABLESPACE DSN8D61A.DSN8S61D
  DELETE DATE(19980910)
```

Chapter 2-12. QUIESCE

The QUIESCE online utility establishes a quiesce point (the current log RBA or log record sequence number (LRSN)) for a table space, partition, table space set, or list of table spaces and table space sets, and records it 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 control statement." For detailed guidance on running this utility, see "Instructions for running QUIESCE" on page 203 .

Output: QUIESCE writes changed pages from the table spaces to DASD. The catalog table SYSIBM.SYSCOPY records the current RBA and the timestamp of the quiesce point. A row with ICTYPE='Q' is inserted into SYSCOPY for each table space quiesced. DB2 also inserts a SYSCOPY row with ICTYPE='Q' for any indexes (defined with the COPY YES attribute) over a table space 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, the privilege set of the process must include one of the following:

- 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.

Execution phases of QUIESCE: The QUIESCE utility operates in these phases:

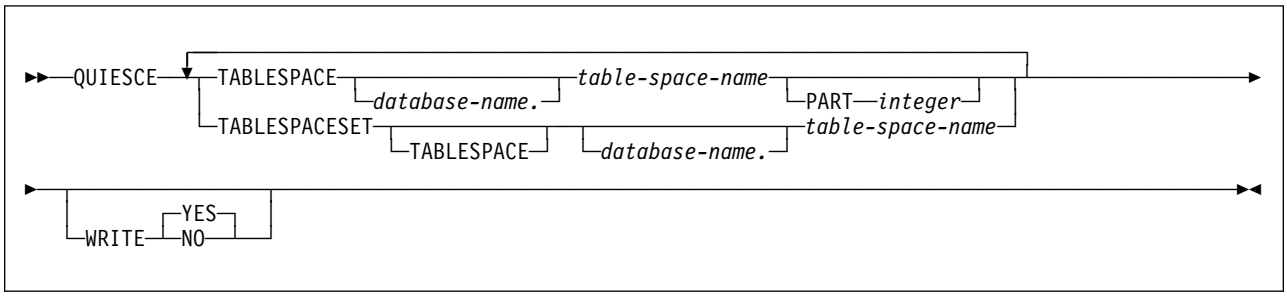
Phase	Description
UTILINIT	Initialization and setup
QUIESCE	Determining the quiesce point and updating the catalog
UTILTERM	Cleanup

Syntax and options of the control statement

The utility control statement defines the function 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

For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.



Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see “Control statement coding rules” on page 27.

TABLESPACE *database-name.table-space-name*
For QUIESCE TABLESPACE, specifies the table space (and, optionally, the database to which it belongs) being quiesced.

For QUIESCE TABLESPACESET, specifies a table space (and, optionally, the database to which it belongs) in the table space set being quiesced.

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 to be quiesced. You can specify DSNDB01.SYSUTILX, but it cannot be included in a list with other table spaces to be quiesced.

PART *integer* Identifies a partition to be quiesced.

integer is the number of the partition and must be in the range from 1 to the number of partitions defined for the table space. The maximum is 254.

TABLESPACESET
Specifies 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 either:

- A group of table spaces that have a referential relationship.
- A base table space with all of its LOB table spaces.

TABLESPACE *database-name.table-space-name*
Specifies the table space name (and, optionally, the database to which it belongs) being quiesced. For QUIESCE TABLESPACESET, the TABLESPACE keyword is optional.

WRITE Specifies whether to write the changed pages from the table spaces and index spaces to DASD.

YES Establishes a quiesce point and writes the changed pages from the table spaces and index spaces to DASD.

The **default** is **YES**.

NO Establishes a quiesce point but does not write the changed pages from the table spaces and index spaces to DASD.

Instructions for running QUIESCE

To run QUIESCE, you must:

1. Read "Before running QUIESCE" in this chapter.
2. Prepare the necessary data sets, as described in "Data sets used by QUIESCE."
3. Create JCL statements, by using one of the methods described in "Chapter 2-1. Invoking DB2 online utilities" on page 27. (For examples of JCL for QUIESCE, see "Sample control statements" on page 207.)
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 204. (For a complete description of the syntax and options for QUIESCE, see "Syntax and options of the control statement" on page 201.)
5. Check the compatibility table in "Concurrency and compatibility" on page 206 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 205.
7. Run QUIESCE.

See "Chapter 2-1. Invoking DB2 online utilities" on page 27 for an explanation of ways to execute DB2 utilities.

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 the COPY pending status" on page 174, "Resetting CHECK pending status" on page 65, "Resetting the REBUILD pending status" on page 175, and Appendix C, "Resetting an advisory or restrictive status" on page 527 for information about resetting these statuses.

Data sets used by QUIESCE

Table 34 on page 204 describes the data sets used by QUIESCE. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 34. Data sets used by QUIESCE

Data Set	Description	Required?
SYSIN	Input data set containing 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 card in the JCL:

Table space
Object to be quiesced. It is named in the QUIESCE control statement and is accessed through the DB2 catalog. (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

See "Syntax and options of the control statement" on page 201 for QUIESCE syntax and option descriptions. See "Sample control statements" on page 207 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"
"Obtaining a common quiesce point"
"Specifying a list of table spaces and table space sets" on page 205

Using QUIESCE for recovery

You can recover a table space to its quiesce point with the RECOVER TABLESPACE utility. See "Chapter 2-14. RECOVER" on page 225 for information about the RECOVER TABLESPACE utility.

Obtaining a common quiesce point

Use the QUIESCE TABLESPACESET utility to obtain a common quiesce point for table spaces that are related. For the purposes of the QUIESCE utility, a table space set is one or both of the following:

- 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 may encounter problems running RECOVER on the table spaces in the structure. RECOVER checks if a complete table space set is recovered to a point in time. If the table space set is not complete, 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 QUIESCE WRITE YES a table space, each related index that was defined with COPY YES has a SYSIBM.SYSCOPY row inserted with ICTYPE='Q' 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 available memory in the batch address space and in the DSN1MSTR address space allows.

Be aware of the following considerations when you specify a list of objects to quiesce:

- Each table space set will be expanded into a list of table spaces that have a referential relationship, or a list containing 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 will continue and the table space will only be quiesced once. QUIESCE will issue 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.

Considerations for running QUIESCE

If a table space is in a pending status: If you run QUIESCE on a table space in COPY pending, CHECK pending, or RECOVER pending status, it terminates as shown in Figure 14.

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = R92341Q
DSBY050I    DSNUGUTC - QUIESCE TABLESPACE UTQPD22A.UTQPS22D
                               TABLESPACE UTQPD22A.UTQPS22E
                               TABLESPACE UTQPD22A.EMPPROJA
DSNU471I  % DSNUQUIA - TABLESPACE  UTQPD22A.EMPPROJA HAS PENDING STATE
DSNU012I DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

Figure 14. QUIESCE and pending restrictions

If the write to DASD fails: QUIESCE attempts to write pages of each table space to DASD. If any of the following conditions is encountered, the write to DASD 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 above conditions is true, QUIESCE will terminate with a return code of 4 and a DSNU473I warning message. In any case, the QUIESCE point is a valid point of consistency for recovery.

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 the QUIESCE utility, but it starts from the beginning again.

For more guidance in restarting online utilities, see “Restarting an online utility” on page 48.

Concurrency and compatibility

Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions of the same table space or index space are compatible.

Table 35 shows which claim classes QUIESCE drains and any restrictive state the utility sets on the target object.

Table 36 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 is also shown.

Table 35. Claim classes of QUIESCE operations. Use of claims and drains; restrictive states set on the target object.

Target	WRITE YES	WRITE NO
Table space or partition	DW/UTRO	DW/UTRO
Index or partition	DW/UTRO	
Nonpartitioning index	DW/UTRO	

Legend:

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

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

Table 36 (Page 1 of 2). QUIESCE compatibility

Action	QUIESCE
CHECK DATA	No
CHECK INDEX	Yes
CHECK LOB	Yes
COPY INDEXSPACE SHRLEVEL REFERENCE	Yes
COPY INDEXSPACE SHRLEVEL CHANGE	No
COPY TABLESPACE SHRLEVEL REFERENCE	Yes
COPY TABLESPACE SHRLEVEL CHANGE	No
DIAGNOSE	Yes
LOAD	No
MERGECOPY	Yes
MODIFY	Yes
QUIESCE	Yes
REBUILD INDEX	No
RECOVER INDEX	No

Table 36 (Page 2 of 2). QUIESCE compatibility

Action	QUIESCE
RECOVER TABLESPACE	No
REORG INDEX	No
REORG TABLESPACE UNLOAD CONTINUE or PAUSE	No
REORG TABLESPACE UNLOAD ONLY or EXTERNAL	Yes
REPAIR DUMP or VERIFY	Yes
REPAIR DELETE or REPLACE	No
REPORT	Yes
RUNSTATS	Yes
STOSPACE	Yes

To run on DSNDB01.SYSUTILX, QUIESCE must be 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 control statements

Example 1: Sample JCL for QUIESCE. Establish a quiesce point for three table spaces.

```
//STEP1    EXEC DSNUPROC,UID='IUIQU2UD.QUIESC2',
//            UTPROC='',SYSTEM='V61A'
//SYSIN    DD *
QUIESCE TABLESPACE DSN8D61A.DSN8S61D
        TABLESPACE DSN8D61A.DSN8S61E
        TABLESPACE DSN8D61A.DSN8S61P
//*
```

Example 2: Sample control statement for QUIESCE. Establish a quiesce point for the DSN8D61A.DSN8S61E and DSN8D61A.DSN8S61D table spaces.

```
QUIESCE TABLESPACE DSN8D61A.DSN8S61E TABLESPACE DSN8D61A.DSN8S61D
```

The following is output of the preceding command:

```
DSNU000I   DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TSLQ.STEP1
DSNU050I   DSNUGUTC - QUIESCE TABLESPACE DSN8D61A.DSN8S61E
                              TABLESPACE DSN8D61A.DSN8S61D
DSNU477I - DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D61A.DSN8S61E
DSNU477I - DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D61A.DSN8S61D
DSNU474I - DSNUQUIA - QUIESCE AT RBA 000000052708
DSNU475I   DSNUQUIB - QUIESCE UTILITY COMPLETE, ELAPSED TIME= 00:00:25
DSNU010I   DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Example 3: QUIESCE point for a table space set. Establish a quiesce point for the table space set of the sample application.

```
QUIESCE TABLESPACESET TABLESPACE DSN8D61A.DSN8S61D
```

The following is output of the preceding command:

```
DSNU000I   DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TSLQ.STEP1
DSNU050I   DSNUGUTC - QUIESCE TABLESPACESET TABLESPACE DSN8D61A.DSN8S61D
DSNU477I - DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACESET DSN8D61A.DSN8S61D
DSNU477I - DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D61A.DSN8S61D
DSNU477I - DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D61A.DSN8S61E
DSNU477I - DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D61A.PROJ
DSNU477I - DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D61A.ACT
DSNU477I - DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D61A.PROJACT
DSNU477I - DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D61A.EMPPROJA
DSNU477I - DSNUQUIA -    QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D61A.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
```

Chapter 2-13. REBUILD INDEX

REBUILD INDEX reconstructs indexes from the table that they reference.

You must modify all RECOVER INDEX jobs from a previous release of DB2 to use REBUILD INDEX instead.

For a diagram of REBUILD INDEX syntax and a description of available options, see "Syntax and options of the control statement." For detailed guidance on running this utility, see "Instructions for running REBUILD INDEX" on page 213

Authorization required: To execute this utility, the privilege set of the process must include one of the following:

- RECOVERDB privilege for the database
- DBADM or DBCTRL authority for the database
- SYSCTRL or SYSADM authority.

To run REBUILD INDEX STATISTICS REPORT YES, the privilege set must include the SELECT privilege on the catalog tables.

Execution phases of REBUILD INDEX: The REBUILD INDEX utility operates in these phases:

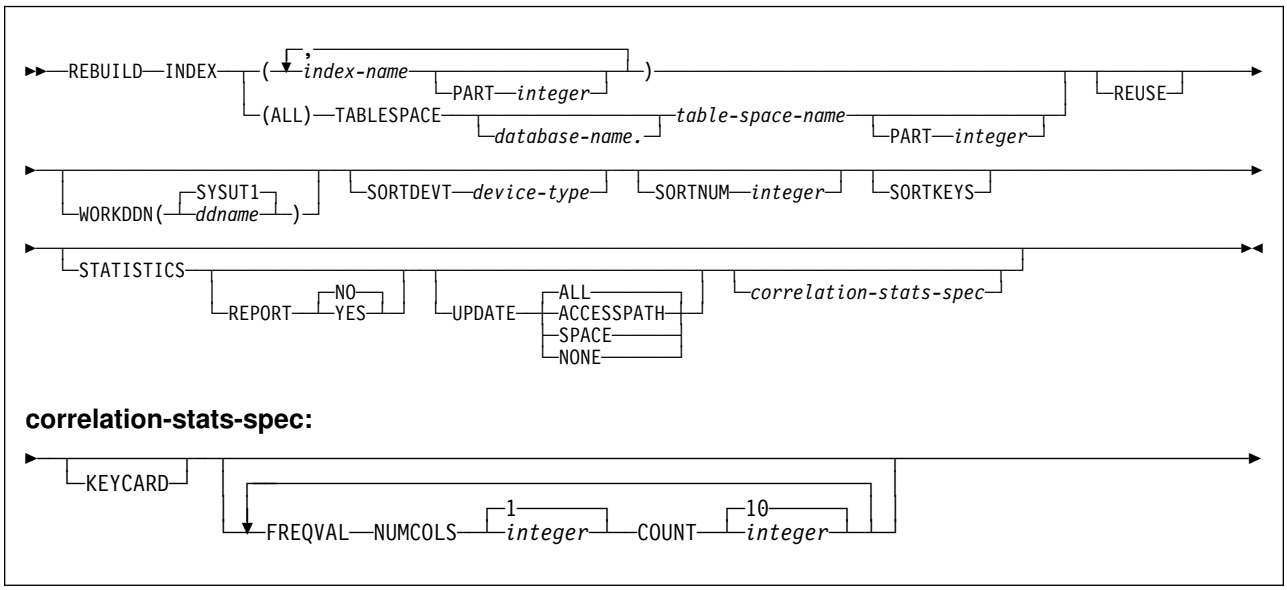
Phase	Description
UTILINIT	Initialization and setup
UNLOAD	Unloading of index entries
SORT	Sorting of unloaded index entries
BUILD	Building of indexes
SORTBLD	If you specify the SORTKEYS keyword to invoke parallel index build processing for a simple or segmented table space, or a single partition of a partitioned table space, all activities that normally occur in both the SORT and BUILD phases occur in the SORTBLD phase instead.
UTILTERM	Cleanup

Syntax and options of the control statement

The utility control statement defines the function 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

For guidance in interpreting syntax diagrams, see "How to read the syntax diagrams" on page 4.



Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see "Control statement coding rules" on page 27.

(*index-name***, ...)** Indicates the qualified name of an index, in the form *creator-id.index-name*. If you omit the qualifier creator ID, the user identifier for the utility job is used.

index-name identifies the index to be rebuilt. To rebuild multiple indexes, separate each index name with a comma. All indexes listed must reside in the same table space. If more than one index is listed and TABLESPACE keyword is not specified, DB2 locates the first valid index name 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 listed.

(ALL) Specifies that all indexes in the table space 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 rebuilt.

PART *integer* Specifies the physical partition of a partitioning index or the logical partition of a nonpartitioning index in a partitioned table space that is to be rebuilt.

integer is the number of the partition and must be in the range from 1 to the number of partitions defined for the table space. The maximum is 254.

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 will not be released if you use the REUSE parameter.

WORKDDN *ddname*

Specifies the DD statement for the temporary work file.

ddname is the DD name for the optional temporary work file.

The **default** is **WORKDDN SYSUT1**. If WORKDDN is omitted and a DD card for SYSUT1 is not provided, REBUILD INDEX performance will improve by eliminating I/O for SORT.

SORTDEVT *device-type*

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

device-type is the device type.

SORTNUM *integer*

Specifies the number of temporary data sets 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. It is allowed to take its own default.

integer is the number of temporary data sets.

SORTKEYS Specifies that index keys are to be sorted and built in parallel during the SORTBLD phase to improve performance, unless constrained by available memory, sort work files, or UTPRIN*nn* file allocations. If you specify SORTKEYS, utility processing ignores any WORKDDN specification or file allocation to SYSUT1 (the default WORKDDN).

STATISTICS Specifies the gathering of index statistics.

If you specify the STATISTICS and UPDATE options, statistics are stored in the DB2 catalog.

REPORT Determines if a set of messages is generated to report the collected statistics.

NO Indicates that the set of messages is not output to SYSPRINT.

The **default** is **REPORT NO**.

YES Indicates that the set of messages is output to SYSPRINT. The messages generated are dependent on the combination of keywords (such as TABLESPACE, INDEX, TABLE, and COLUMN) 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 Tells whether the collected statistics are inserted into the catalog tables. UPDATE also allows you to select statistics used for access path selection or statistics used by database administrators.

ALL Indicates that all collected statistics will be updated in the catalog.

The **default** is **UPDATE ALL**.

ACCESSPATH Indicates that only the catalog table columns that provide statistics used for access path selection are updated.

SPACE Indicates that only the catalog table columns that provide statistics to help the database administrator assess the status of a particular table space or index are updated.

NONE Indicates that no catalog tables are updated with the collected statistics. This option is only valid when REPORT YES is specified.

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.

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 to concatenate together when collecting frequent values from the specified index. Specifying '3' means to collect frequent values on the concatenation of the first three key columns. The default is 1, which means collect frequent values on the first key column of the index.

COUNT Indicates the number of frequent values to be collected. Specifying '15' means collect 15 frequent values from the specified key columns. The default is *10*.

Instructions for running REBUILD INDEX

To run REBUILD INDEX, you must:

1. Read “Before running REBUILD INDEX” in this chapter.
2. Prepare the necessary data sets, as described in “Data sets used by REBUILD INDEX.”
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27. (For examples of JCL for REBUILD INDEX, see “Sample control statements” on page 221.)
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 214. (For a complete description of the syntax and options for REBUILD INDEX, see “Syntax and options of the control statement” on page 209.)
5. Check the compatibility table in “Concurrency and compatibility” on page 220 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 219.
7. Run REBUILD INDEX.

See “Chapter 2-1. Invoking DB2 online utilities” on page 27 for an explanation of ways to execute DB2 utilities.

Before running REBUILD INDEX

Because the data needed 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 also is damaged. Neither do you have to rebuild an index merely because you have recovered the table space it is based on.

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.

Data sets used by REBUILD INDEX

Table 37 describes the data sets used by REBUILD INDEX. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 37 (Page 1 of 2). Data sets used by REBUILD INDEX

Data Set	Description	Required?
SYSIN	Input data set containing the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
Work data sets	Temporary data sets for sort input and output. The DD names have the form SORTWK*nn*.	Yes

Table 37 (Page 2 of 2). Data sets used by REBUILD INDEX

Data Set	Description	Required?
Work data set	Temporary data set used to store the index keys for REBUILD INDEX. The WORKDDN option of the utility control statement specifies its DD name. The default DD name is SYSUT1. To find the approximate size in bytes of the work data set, see page 214.	No

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

Table space
Object to be rebuilt. It is named in the REBUILD INDEX control statement and is accessed through the DB2 catalog.

Creating the work data set: REBUILD INDEX can use a single sequential data set as described by the DD statement specified in the WORKDDN option.

To calculate the approximate size (in bytes) of the WORKDDN data set, follow these steps:

1. For each table, multiply the number of records in the table by the number of indexes needing to be rebuilt on the table.
2. Add the products obtained in step 1.
3. Multiply the sum (from step 2) by the longest key length plus 9.

Allocating twice the space 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.

Creating the control statement

See “Syntax and options of the control statement” on page 209 for syntax and option descriptions. See “Sample control statements” on page 221 for examples of usage.

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 215
“Improving performance and space utilization” on page 215
“Building indexes in parallel for REBUILD INDEX” on page 215
“Resetting the REBUILD pending status” on page 218
“Rebuilding critical catalog indexes” on page 219
“Rebuilt index recoverability” on page 219

Rebuilding index partitions

REBUILD INDEX can rebuild one or more partitions of a partitioning index. The PART option allows you to specify a particular partition to be rebuilt. This prevents REBUILD INDEX from unnecessarily scanning the entire table space, and unnecessarily rebuilding every index. If you recover any part of a partitioned table space to the latest point of consistency, you must rebuild all nonpartitioning indexes.

Improving performance and space utilization

REBUILD INDEX rebuilds indexes by re-creating them from the tables on which they are based. It can rebuild one or more partitions of a partitioning index in a partitioned table space when you specify the PART option. This prevents REBUILD INDEX from unnecessarily scanning the entire table space when rebuilding only a single partition of an index.

To rebuild several indexes simultaneously and reduce recovery time, use parallel index rebuild, or submit multiple index jobs. See “Building indexes in parallel for REBUILD INDEX” for more information.

When rebuilding nonpartitioning indexes and partitions of partitioning 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 required to sort all the keys.

When you run the REBUILD INDEX utility concurrently on separate partitions of a partitioning index, the sum of the processor time will be roughly equivalent to the time it takes to run a single REBUILD INDEX job against the entire index. For partitioning indexes, the elapsed time for running concurrent REBUILD INDEX jobs will be a fraction of the elapsed time for running a single REBUILD INDEX job against an entire index.

REBUILD INDEX utility performance can be improved by eliminating the work data set; however, if the job terminates abnormally, you will have to restart it from the beginning.

Building indexes in parallel for REBUILD INDEX

Use parallel index build to reduce the elapsed time for a REBUILD INDEX job by sorting the index keys and rebuilding multiple indexes 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.

The greatest elapsed processing time improvements result from parallel rebuilding for:

- Multiple indexes on a simple, segmented, or partitioned table space
- The partitioning index on all partitions of a partitioned table space
- The nonpartitioning index on a partitioned table space

Figure 15 on page 216 shows a REBUILD INDEX flow with parallel index build, which requires SORTKEYS. DB2 starts multiple subtasks to unload all parts of the partitioned table space. Subtasks then sort index keys and build the partitioning index in parallel. If you specified 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.

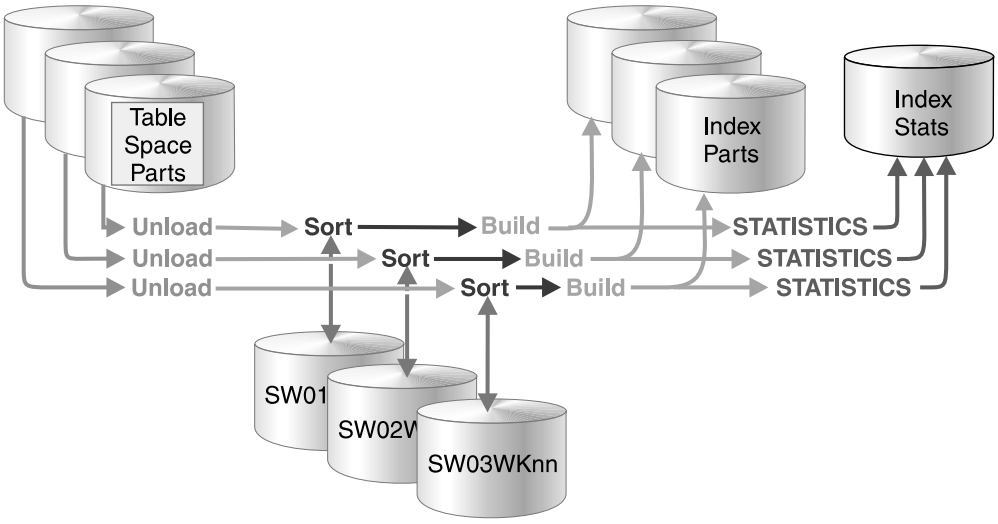


Figure 15. Rebuilding a partitioning index using parallel index build

Figure 16 shows a REBUILD INDEX flow with parallel index build, which requires SORTKEYS. DB2 starts multiple subtasks to unload all partitions of a partitioned table space, and sort index keys in parallel. The keys are then merged and passed to the build subtask, which builds the nonpartitioning index. If you specified STATISTICS, a separate subtask collects the sorted keys and updates the catalog table.

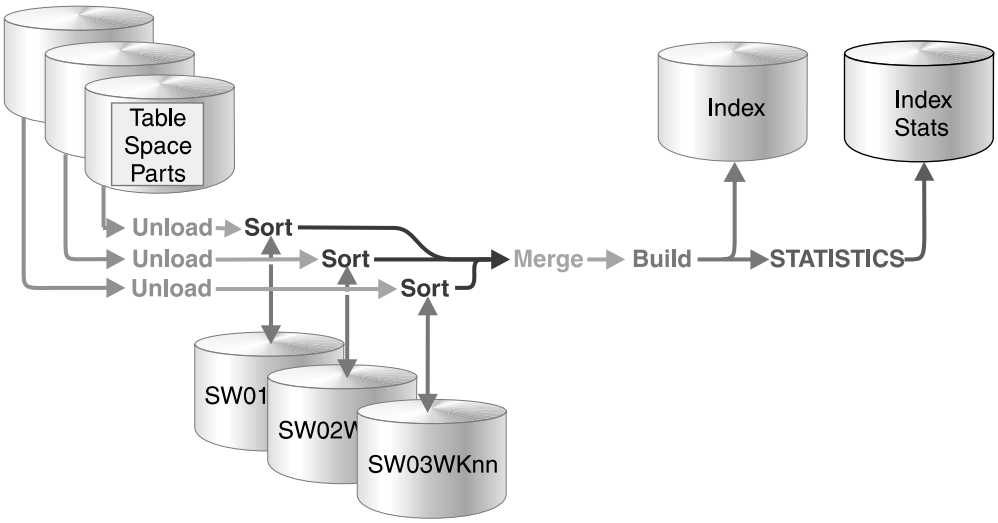


Figure 16. Rebuilding a nonpartitioning index using parallel index build

REBUILD INDEX of a table space or partition uses parallel index build if all of the following conditions are true:

- You specify the SORTKEYS keyword in the utility statement.
- You either allow the utility to dynamically allocate the data sets needed by SORT, or provide the necessary data sets yourself.

Select one of the following methods to allocate sort work and message data sets:

Method 1: REBUILD INDEX determines the optimal number of sort work and message data sets.

1. Specify the SORTKEYS and SORTDEVT keywords 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. Specify the SORTKEYS keyword in the utility statement.
2. Provide DD statements with DDNAMEs in the form SW*nn*WK*mm*.
3. 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. Specify the SORTKEYS keyword in the utility statement.
2. Provide DD statements with DDNAMEs in the form SW*nn*WK*mm*.
3. Provide DD statements with DDNAMEs in the form UTPRIN*nn*.

Data sets used: If you select Method 2 or 3 above, define the necessary data sets by using the information provided here, along with:

- "Determining the number of sort subtasks" on page 218
- "Allocation of sort subtasks" on page 218
- "Estimating the sort work file size" on page 218

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 nonpartitioning 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 DASD free space
- Limit the number of utility subtasks used to build indexes

The DDNAMEs 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 building the first index.

SW01WK02 The second sort work data set that is used by the subtask building the first index.

SW02WK01 The first sort work data set that is used by the subtask building the second index.

SW02WK02 The second sort work data set that is used by the subtask building the second index.

The DDNAMEs 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, 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 do not supply 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 started for parallel index build is equal to the number of indexes that are to be built for a simple table space, segmented table space, or single partition of a partitioned table space, or the number of partitions that are to be unloaded if only a single index is being built on a partitioned table space.

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 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 being processed by the subtask in order to calculate each sort work file size. When you've determined which indexes are assigned to which subtask pairs, use the following formula to calculate the space required:

2 × (longest index key + 9) × (number of keys extracted)

longest key The length of the longest index key that is to be processed by the subtask. For the first subtask pair for REBUILD INDEX, use the maximum value of the longest key.

number of keys The number of keys from all indexes to be sorted that are to be processed by the subtask.

Resetting the REBUILD pending status

REBUILD pending (RBDP in DISPLAY command output) means that the physical or logical index partition, nonpartitioning index, or logical partition of a nonpartitioning index is in REBUILD pending status.

There are three variations of REBUILD pending:

RBDP The physical or logical index partition is in the REBUILD pending status. The individual physical or logical index partition is inaccessible. RBDP is reset by rebuilding the single affected partition.

RBDP* The logical partition of the nonpartitioning index is in the REBUILD pending status. The entire nonpartitioning index is inaccessible. RBDP* is reset by rebuilding only the affected logical partitions.

PSRBD The nonpartitioning index space is in the REBUILD pending status. The entire index space is inaccessible and must be rebuilt with the REBUILD INDEX utility.

You can reset the REBUILD pending status for an index with any of these operations:

- REBUILD INDEX
- REORG INDEX SORTDATA
- REPAIR SET INDEX with NORBDPEND
- -START DATABASE with ACCESS FORCE

Rebuilding critical catalog indexes

An ID with a granted authority receives message DSNT500I, "RESOURCE UNAVAILABLE," while trying to rebuild indexes in the catalog or directory if the DSNDB06.SYSDBASE or DSNDB06.SYSUSER table space is unavailable. If you get this message, 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.

Rebuilt index recoverability

When you successfully rebuild an index that was defined with COPY YES, utility processing inserts a SYSIBM.SYSCOPY row with ICTYPE='B' for each index rebuilt. Rebuilt indexes are also placed in informational COPY pending status. We recommend taking a full image copy of the index to create a recoverable point in time; this action also resets the ICOPY status.

Terminating or restarting REBUILD INDEX

You can terminate REBUILD INDEX with the TERM UTILITY command. If you terminate a REBUILD INDEX job, the table space is placed in the RECOVER pending status and is unavailable until it has been successfully rebuilt.

If you specified the WORKDDN keyword, you can restart REBUILD INDEX during the UNLOAD and SORT phases, and at the last index built during the BUILD phase. However, there is a short period of time during writing of SORT output at the end of the SORT phase that requires restart to begin at the beginning of the UNLOAD phase instead of at the SORT phase. If you omit WORKDDN, the job starts over again from the beginning.

If you restart a job that used the SORTKEYS keyword, you must restart from the beginning of the UNLOAD phase.

If you restart a job which uses the STATISTICS keyword, inline statistics collection will not occur. To update catalog statistics, run the RUNSTATS utility after the restarted REBUILD INDEX job completes.

For more guidance in restarting online utilities, see "Restarting an online utility" on page 48.

Concurrency and compatibility

Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions of the same table space or index space are compatible.

Table 38 shows which claim classes REBUILD INDEX drains and any restrictive state the utility sets on the target object.

Table 39 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 is also shown.

Table 38. Claim classes of REBUILD INDEX operations. Use of claims and drains; restrictive states set on the target object.

Target	REBUILD INDEX	REBUILD INDEX PART
Table space or partition	DW/UTRO	DW/UTRO
Index or physical partition	DA/UTUT	DA/UTUT
Nonpartitioning index	DA/UTUT	DR
Logical partition of an index		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 “RR” claim class
- UTUT - Utility restrictive state - exclusive control
- UTRO - Utility restrictive state - read only access allowed.

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

Table 39 (Page 1 of 2). REBUILD INDEX compatibility

Action	REBUILD INDEX
CHECK DATA	No
CHECK INDEX	No
CHECK LOB	Yes
COPY INDEX	No
COPY TABLESPACE SHRLEVEL REFERENCE	Yes
COPY TABLESPACE SHRLEVEL CHANGE	No
DIAGNOSE	Yes
LOAD	No
MERGECOPY	Yes
MODIFY	Yes
QUIESCE	No
REBUILD INDEX	No

Table 39 (Page 2 of 2). REBUILD INDEX compatibility

Action	REBUILD INDEX
RECOVER INDEX	No
RECOVER TABLESPACE	No
REORG INDEX	No
REORG TABLESPACE UNLOAD CONTINUE or PAUSE	No
REORG TABLESPACE UNLOAD ONLY or EXTERNAL without cluster index	Yes
REORG TABLESPACE UNLOAD ONLY or EXTERNAL with cluster index	No
REPAIR LOCATE by KEY	No
REPAIR LOCATE by RID DUMP or VERIFY	Yes
REPAIR LOCATE by RID DELETE or REPLACE	No
REPAIR LOCATE TABLESPACE PAGE DUMP or VERIFY	Yes
REPAIR LOCATE INDEX PAGE DUMP or VERIFY	No
REPAIR LOCATE TABLESPACE or INDEX PAGE REPLACE	No
REPORT	Yes
RUNSTATS INDEX	No
RUNSTATS TABLESPACE	Yes
STOSPACE	Yes

To run on SYSIBM.DSNLUX01 or SYSIBM.DSNLUX02, REBUILD INDEX must be the only utility in the job step and the only utility running in the DB2 subsystem.

Sample control statements

Example 1: Rebuild an index. Rebuild the DSN8610.XDEPT1 index, which indexes the DSN8610.TDEPT table in the DSN8D61A database.

```
//STEP1    EXEC DSNUPROC,UID='IUIQU2UT.RBLD1',TIME=1440,
//         UTPROC='',
//         SYSTEM='V61A',DB2LEV=DB2A
//SYSREC   DD DSN=IUIQU2UT.RBLD1.STEP1.SYSREC,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(8000,(20,20),,,ROUND)
//SYSUT1   DD DSN=IUIQU2UT.RBLD1.STEP1.SYSUT1,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(8000,(20,20),,,ROUND)
//SYSIN    DD *
REBUILD INDEX (DSN8610.XDEPT1)
//*
```

Example 2: Rebuild index partitions. Rebuild partitions 2 and 3 of the DSN8610.XEMP1 index.

```
REBUILD INDEX (DSN8610.XEMP1 PART 2, DSN8610.XEMP1 PART 3)
```

Example 3: Rebuild a single index on a segmented table space. Rebuild the DSN8610.XDEPT1 index. This example specifies the SORTKEYS keyword to use parallelism and uses dynamic data set and message set allocation with the SORTDEVT and SORTNUM keywords.

DB2 starts one utility sort subtask pair to build the 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.

```
//SAMPJOB  JOB  ...
//STEP1    EXEC DSNUPROC,UID='SAMPJOB.RCVINDEX',UTPROC='',SYSTEM='V61A'
//SYSIN    DD *
REBUILD INDEX (DSN8610.XDEPT1)
 SORTDEVT SYSWK
 SORTNUM  4
 SORTKEYS
/*
```

Example 4: Rebuild multiple partitions of a partitioning index. Rebuild partitions 2 and 3 of the DSN8610.XDEPT1 index, using parallel index build processing. This example specifies the SORTKEYS keyword to use parallelism and uses dynamic data set and message set allocation with the SORTDEVT and SORTNUM keywords.

If sufficient virtual storage resources are available, DB2 starts one utility sort subtask pair for each partition. This example does not require UTPRIN*nn* DD statements, because it uses DSNUPROC to invoke utility processing, which includes a DD statement allocating UTPRINT to SYSOUT.

```
//SAMPJOB  JOB  ...
//STEP1    EXEC DSNUPROC,UID='SAMPJOB.RBINDEX',UTPROC='',SYSTEM='V61A'
//SYSIN    DD *
REBUILD INDEX (DSN8610.XEMP1 PART 2, DSN8610.XEMP1 PART 3)
 SORTDEVT SYSWK
 SORTNUM  4
 SORTKEYS
/*
```

Example 5: Rebuild all partitions of a partitioning index. Rebuilds all index partitions of the DSN8610.XEMP1 partitioning index, using parallel index build processing. This example specifies the SORTKEYS keyword and 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, which includes a DD statement allocating UTPRINT to SYSOUT.

```
//SAMPJOB  JOB  ...
//STEP1    EXEC DSNUPROC,UID='SAMPJOB.RCVINDEX',UTPROC='',SYSTEM='V61A'
//* 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 (DSN8610.XEMP1)
   SORTKEYS
/*
```

Example 6: Rebuild all indexes of a partitioned table space. Rebuild all indexes for table space DSN8S61E in database DSN8D61A, using parallel index build processing. This example specifies the SORTKEYS keyword and uses dynamic data set and message set allocation with the SORTDEVT and SORTNUM keywords.

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 non-partitioning index. This example does not require UTPRIN*nn* DD statements, because it uses DSNUPROC to invoke utility processing, which includes a DD statement allocating UTPRINT to SYSOUT.

```
//SAMPJOB  JOB  ...
//STEP1    EXEC DSNUPROC,UID='SAMPJOB.RCVINDEX',UTPROC='',SYSTEM='V61A'
//SYSIN    DD *
REBUILD INDEX (ALL) TABLESPACE DSN8D61A.DSN8S61E
 SORTKEYS
 SORTDEVT SYSWK
 SORTNUM  4
/*
```

Chapter 2-14. RECOVER

The RECOVER online utility recovers data to the current state or to a previous point in time by restoring a copy, 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. RECOVER 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 containing changes to the object. If the most recent full image copy data set is unusable, and there are previous image copy data sets existing in the system, then RECOVER uses the previous image copy data sets. For more information about using data sets for recovery, see “Recovering a data set or partition” on page 235.

Compatibility with prior releases: In previous releases of DB2, REBUILD INDEX was called RECOVER INDEX. You must modify all utility control statements from previous releases to use REBUILD INDEX if you want to continue recovering the indexes via a scan of the data. However, if you want to recover the indexes from a full image copy, change those control statements to use the new RECOVER INDEX syntax on page 226. Only indexes that were defined with the COPY YES attribute can be copied and subsequently recovered; see Chapter 6 of *DB2 SQL Reference* for more information about the COPY YES attribute of the ALTER INDEX and CREATE INDEX SQL statements.

For a diagram of RECOVER syntax and a description of available options, see “Syntax and options of the control statement” on page 226. For detailed guidance on running this utility, see “Instructions for running RECOVER” on page 232.

Output: Output from RECOVER consists of recovered data (either a table space, index, partition or data set, error range, or page within a table space).

If you specify the TOLOGPOINT, TORBA, or TOCOPY 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 index space from REBUILD pending status.

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, including 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 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.

If you use the RECOVER utility to partially recover data and all indexes over the data, it is best to recover these objects to a common quiesce point or SHRLEVEL REFERENCE copy. Otherwise, RECOVER places all indexes in the CHECK pending status.

Authorization required: To execute this utility, the privilege set of the process must include one of the following:

- 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.

Execution phases of RECOVER: The RECOVER utility operates in these phases:

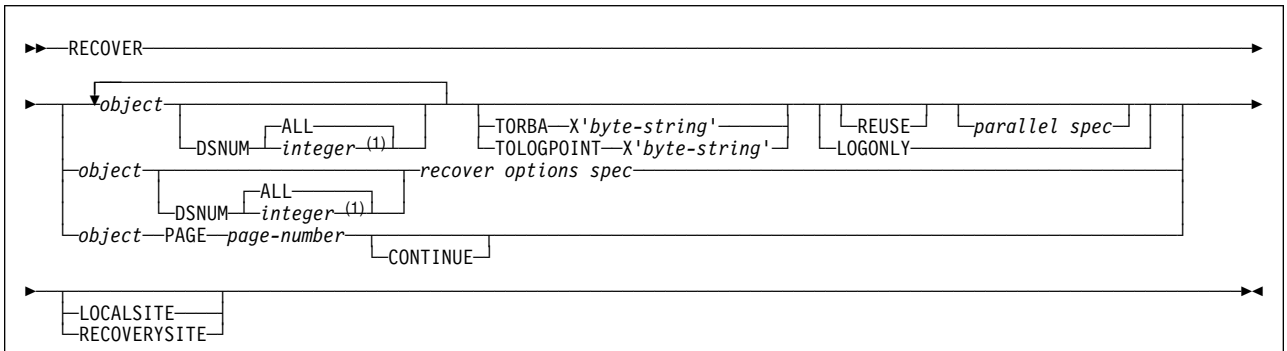
Phase	Description
UTILINIT	Initialization and setup
RESTORE	Locate and merge any appropriate image copies and restore the table space to a backup level; processes a list of objects in parallel if you specified the PARALLEL keyword.
RESTORER	If you specified the PARALLEL keyword, reads and merges the image copies.
RESTOREW	If you specified the PARALLEL keyword, writes the pages to the object.
LOGAPPLY	Apply any outstanding log changes to the object restored from the previous phase or step.
UTILTERM	Cleanup

Syntax and options of the control statement

The utility control statement defines the function 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.

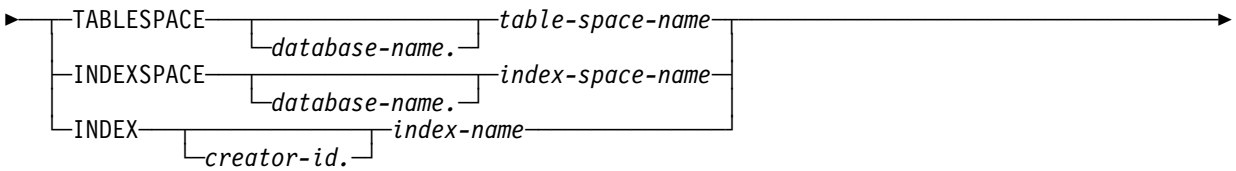
Syntax diagram

For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.

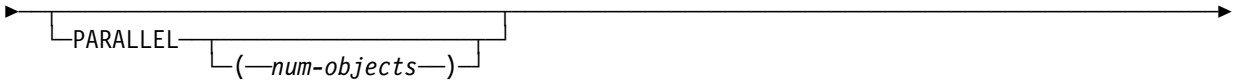


Note:
[1] Not valid for nonpartitioning indexes.

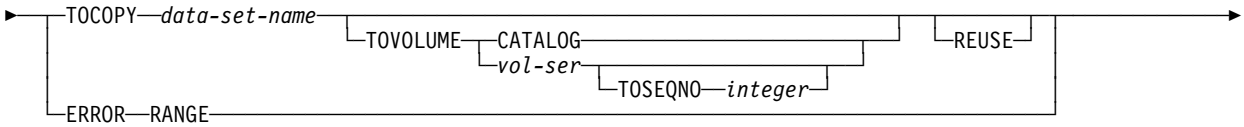
object:



parallel 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.

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.

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see “Control statement coding rules” on page 27.

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 the table space belongs to.

The **default** is **DSNDB04**.

table-space-name Is the name of the table space 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 the index space belongs to.

The **default** is **DSNDB04**.

index-space-name Specifies the name of the index space 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. For more information about this restriction, see "Compatibility with Prior Releases" on page 225.

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 to be recovered.

The following are optional:

DSNUM Identifies a partition or data set within a table space, or a partition within an index space, that is to be recovered; or it recovers the entire table space or index space.

ALL Recovers the entire table space or index space.

The **default** is **ALL**.

integer Is the number of a partition or data set to be recovered. The maximum is 254.

Not valid for non-partitioning 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 as cataloged in the VSAM catalog. The data set name has this format:

catname.DSNDB*x*.*dbname*.*tsname*.I0001.A*nnn*

where:

catname The VSAM catalog name or alias
x C or D
dbname The database name
tsname The table space name
nnn The data set integer.

PAGE *page-number*

Specifies a particular page to be recovered.

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.

CONTINUE Specifies that the recovery process is to continue. Use this option only if RECOVER has terminated during reconstruction of a page, because of an error. In this case, the page is marked as "broken." After you have repaired 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***'**

Is used in a non-data-sharing environment to specify a point on the log to recover to. You must specify TORBA when recovering to a point before Version 4. Specify an RBA value.

In a data sharing environment, TORBA must be used only when recovering to a point before the originating member joined the data sharing group. If you specify an RBA after this point, the recovery fails.

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 recover to. Specify either an RBA or an LRSN value.

The LRSN is a string of 12 hexadecimal characters and is reported by the DSN1LOGP utility. The value must be greater than the LRSN value when Version 4 started.

LOGONLY Recovers the target objects 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.

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 DFSMS Concurrent Copy.

REUSE Specifies that RECOVER logically resets and reuses 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 will not be released if you use the REUSE parameter.

PARALLEL Specifies the maximum number of objects in the list that should be restored in parallel from image copies on DASD. Specify the PARALLEL keyword to take advantage of parallel processing during the RESTORE phase.

(*num-objects***)** Specifies the number of objects in the list that should 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.

LOCALSITE
RECOVER uses image copies from the local site. If you specify neither LOCALSITE or RECOVERYSITE, then 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
RECOVER uses image copies from the recovery site. If you specify neither LOCALSITE or RECOVERYSITE, then 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.)

TOCOPY *data-set-name*
Specifies the particular image copy data set that DB2 uses 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 used in recovery. If it is an incremental image copy, the recovery 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), then 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.

If the image copy data set is an MVS 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 there is more than one image copy data set with the same data set name, use one of the following options to identify the data set exactly:

TOVOLUME
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.)

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 stored on multiple tape volumes.

TOSEQNO *integer*
Identifies the image copy data set by its file sequence number.

integer is the file sequence number.

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 containing it; otherwise, it is probably better to recover the entire object.

There are some situations in which 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. The IBM Device Support Facility, ICKDSF service utility, can be used to determine whether this situation exists. In such a situation, the error data set should be redefined at a different location on the volume or on a different volume and the RECOVER utility will run without the ERROR RANGE option.

Refer to Section 4 (Volume 1) of *DB2 Administration Guide* for additional information concerning the use of this keyword.

Instructions for running RECOVER

To run RECOVER, you must:

1. Read “Before running RECOVER” in this chapter.
2. Prepare the necessary data sets, as described in “Data sets used by RECOVER” on page 233.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27. (For examples of JCL for RECOVER, see “Sample control statements” on page 252.)
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 233. (For a complete description of the syntax and options for RECOVER, see “Syntax and options of the control statement” on page 226.)
5. Check the compatibility table in “Concurrency and compatibility” on page 250 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 250.
7. Run RECOVER.

See “Chapter 2-1. Invoking DB2 online utilities” on page 27 for an explanation of ways to execute DB2 utilities.

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, then 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 defines 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 used by RECOVER

Table 40 describes the data sets used by RECOVER. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 40. Data sets used by RECOVER

Data Set	Description	Required?
SYSIN	Input data set containing 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 cards in the JCL:

Table space or Index space The name of the table space or index space to be recovered. It is named in the control statement and is accessed through the DB2 catalog. 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 This information is accessed through the DB2 catalog. However, if you would like to preallocate your image copy data sets by using DD cards, refer to “Retaining tape mounts” on page 249 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 recovers table space DSN8S61D in database DSN8D61A:

```
RECOVER TABLESPACE DSN8D61A.DSN8S61D
```

You can also recover multiple table spaces by creating a list of table spaces to be recovered; repeat the TABLESPACE keyword before each table space specified. The following statement recovers partition 2 of the partitioned table space DSN8D61A.DSN8S61E, and recovers the table space DSN8D61A.DSN8S61D to the quiesce point (`RBA X'000007425468'`).

```
RECOVER TABLESPACE DSN8D61A.DSN8S61E DSNUM 2
        TABLESPACE DSN8D61A.DSN8S61D
        TORBA X'000007425468'
```

Each table space 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 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 separate RECOVER operations. The following example shows the RECOVER statement for recovering four data sets in database DSN8D61A, table space DSN8S61E:

```
RECOVER TABLESPACE DSN8D61A.DSN8S61E DSNUM 1
        TABLESPACE DSN8D61A.DSN8S61E DSNUM 2
        TABLESPACE DSN8D61A.DSN8S61E DSNUM 3
        TABLESPACE DSN8D61A.DSN8S61E DSNUM 4
```

The recovery of these data sets can be scheduled 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. This status can be reset in several ways: see “Resetting the COPY pending status” on page 174.

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 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 get a table space listing.

RECOVER 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 for image copies on DASD devices. If RECOVER encounters a tape volume in the list, processing of remaining objects pauses until the tape object has completed, and then parallel processing resumes.

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.

When image copies are done at the data set level, then RECOVER must be done at the data set level. To recover the whole table space, all the data sets must be recovered individually in one or more RECOVER steps. If recovery is attempted at the table space level, the following message is received:

```
DSNU514I DSNUCASA - RECOVERY DATA DOES NOT PERMIT
                    TABLESPACE RECOVERY
```

Alternatively, if image copies are taken at the table space, index, or index space level, individual data sets can be recovered simply by coding the DSNUM parameter.

RECOVER does *not* support recovery of:

- A single data set for non-partitioning indexes
- A logical part of a nonpartitioning index

You should consider the effects of data compression on recovery. When you use the option TOCOPY, TOLOGPOINT, or TORBA to recover a single data set of a nonpartitioned table space, message DSNU520I is issued to warn you that the table space might be inconsistent after the RECOVER job. This point-in-time recovery might cause compressed data to exist without a dictionary or could even overwrite the data set that contains the current dictionary.

Recovering with incremental copies

The RECOVER utility merges all incremental image copies since the last full image copy, and must have all the image copies available at the same time. If there is any likelihood that the requirement will 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 there are not enough tape units, the utility can still perform. RECOVER dynamically allocates the full image copy and attempts to allocate dynamically 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 simply ignored.

Recovering a page

RECOVER PAGE allows you to recover data on a page that has been 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 using PAGE and CONTINUE: Suppose 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. Changing data to invalid values using REPLACE can produce unpredictable results, particularly when changing page header information. Improper use of REPAIR can result in damaged data, or in some cases, system failure.

2. Resubmit the RECOVER utility job specifying TABLESPACE(TSPACE1) PAGE(*5*) CONTINUE. The RECOVER utility finishes recovering the damaged page by applying the log records remaining after the one that caused the problem.

If more than one page is damaged during RECOVER, do step 1 and step 2 for each damaged page.

Recovering an error range

The ERROR RANGE option of RECOVER allows you to 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 statement recovers 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, it is probably better to recover the entire object.

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 DASD error. If an I/O error is detected during RECOVER processing, DB2 issues message DSNU538I to tell you which target tracks are involved. The message provides enough information to run ICKDSF correctly.

There are some situations, announced by error messages, in which recovery of an error range only is not possible. In such a situation, it is better to recover the entire object.

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, then you can remove the bad volume from the storage group using ALTER STOGROUP.

Recovering with a data set copy 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 will cause RECOVER to skip the RESTORE phase and apply the log records only, starting from the first log record written after the data set was backed up.

Because the data sets are restored offline without DB2's involvement, RECOVER LOGONLY checks that the data set identifiers match those in the DB2 catalog. If they 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:

1. Stop the DB2 objects 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 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 the LOGONLY parameters.
5. If you did not recover related indexes in the same RECOVER control statement, then 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™, then 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. It 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 must recover SYSLGRNX, SYSUTILX, and SYSUSER, recover first SYSUTILX, then SYSLGRNX, then SYSUSER. It is not necessary to recover all of the objects, only those that require recovery.

1. DSNDB01.SYSUTILX
2. All indexes on SYSUTILX
3. DSNDB01.DBD01
4. DSNDB06.SYSCOPY
5. All IBM defined indexes on SYSCOPY[3]
6. DSNDB01.SYSLGRNX
7. All indexes on SYSLGRNX
8. DSNDB06.SYSDBAUT
9. All IBM defined indexes on SYSDBAUT[3]
10. DSNDB06.SYSUSER
11. DSNDB06.SYSDBASE
12. All IBM defined indexes on SYSDBASE and SYSUSER[3]
13. Other catalog and directory table spaces and indexes. The remaining catalog table spaces, in database DSNDB06, are SYSGROUP, SYSGPAUT, SYSOBJ, SYSPLAN, SYSPKAGE, SYSSEQ, SYSSEQ2, SYSSTATS, SYSSTR, and SYSVIEWS. Most indexes are listed in *DB2 SQL Reference* . One index not listed there is DSNVTH01. There are two remaining directory table spaces, DSNDB01.SCT02, which has indexes SYSIBM.DSNSCT02, and DSNDB01.SPT01, which has indexes SYSIBM.DSNSPT01 and SYSIBM.DSNSPT02.
14. All user defined indexes on the catalog.
15. System utility table spaces such as QMF.
16. If used, the communications database (CDB), the object and application registration tables, and the resource limit specification tables.
17. User table spaces.

For all catalog and directory table spaces, the IBM-defined indexes with the COPY YES attribute can be listed in the same RECOVER utility statement.

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

[3] If there are no user defined indexes on the catalog, execute `REBUILD INDEX (ALL) TABLESPACE DSNDB06.SYSxxxx` to rebuild all the IBM defined indexes on a catalog table space. If user defined indexes are created on the catalog, the IBM defined indexes must be rebuilt individually and the user defined indexes rebuilt in a subsequent step. See Appendix D of *DB2 SQL Reference* for a list of the IBM defined indexes.

2. When you recover the following table spaces, no other utilities can run while the RECOVER utility is running. Other utility statements may exist in the same job step.
   - DSNDB06.SYSCOPY
   - DSNDB01.SYSLGRNX
   - DSNDB06.SYSDBAUT
   - DSNDB06.SYSUSER
   - DSNDB06.SYSDBASE

Attention: 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 on recovering the BSDS, see Section 4 (Volume 1) of *DB2 Administration Guide* .

The access method services REPRO function should be used to copy active log data sets. For information on the JCL for the access method services REPRO function, see:

- *DFSMS/MVS: Access Method Services for the Integrated Catalog*
- *DFSMS/MVS: Access Method Services for VSAM Catalogs*

Why the order is important: To recover one object, RECOVER must obtain information about it from some other object. Table 41 lists the objects from which RECOVER must obtain information.

Table 41. Objects the RECOVER utility accesses

Object Name	Reason for Access by RECOVER
DSNDB01.SYSUTILX	Utility restart information. It is not accessed when it is recovered; RECOVER for this object is not restartable, and there can be no other commands in the same job step. SYSCOPY information for SYSUTILX is obtained from the log.
DSNDB01.DBD01	Descriptors for the catalog database (DSNDB06), the work file database (DSNDB07), and user databases. RECOVER for this object is not restartable, and there can be no other commands 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	To verify that the authorization ID is authorized to run RECOVER.
DSNDB06.SYSDBASE	Descriptors of table spaces to be recovered.

You can use REPORT RECOVERY to get SYSCOPY information for DSNDB01.SYSUTILX, DSNDB01.DBD01, and DSNDB06.SYSCOPY.

Units of Recovery: When you recover the DB2 catalog and directory, consider the entire catalog and directory as one unit of recovery. Recover all table spaces and index spaces of the catalog and directory to the same point of consistency. Sample

queries and documentation are provided in DSNTESQ in the SDSNSAMP sample library that can be used 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 have 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 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 the indexes over them, 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, then one method of creating a point of consistency is to:

1. Quiesce all catalog and directory table spaces in a list, except for DSNDB06.SYSCOPY and DSNDB01.SYSUTILX.
2. Quiesce DSNDB06.SYSCOPY.

   We recommend that you quiesce DSNDB06.SYSCOPY in a separate utility statement; when you recover DSNDB06.SYSCOPY to its own quiesce point, it contains the ICTYPE = 'Q' (quiesce) SYSIBM.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 on page 239.

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 get this message, you must either make these table spaces available or run the RECOVER utility on the catalog or directory using an authorization ID with the installation SYSADM or installation SYSOPR authority.

Recovering a table space containing 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. A LOB can be marked invalid if all of the following 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 related to a LOB table space can change due to a recovery operation, depending on the type of recovery 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 will exist:

- base table space
- index on the auxiliary table
- LOB table space

Refer to Table 42 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 42. Determining object status after recovery.

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	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	None
LOB table space	Current RBA or LRSN, LOB table space defined with LOG(YES)	None	None	None
LOB table space	Current RBA or LRSN, LOB table space defined with LOG(NO)	None	None	auxiliary warning[1]
LOB table space	TOCOPY, COPY was SHRLEVEL REFERENCE	CHECK pending	REBUILD pending	None
LOB table space	TOCOPY, COPY was SHRLEVEL CHANGE	CHECK pending	REBUILD pending	CHECK pending-auxiliary warning[1]
LOB table space	TOLOGPOINT or TORBA (not a quiesce point)	CHECK pending	REBUILD pending	CHECK pending-auxiliary warning[1]
LOB table space	TOLOGPOINT or TORBA (at a quiesce point)	CHECK pending	REBUILD pending	None

Note: [1] If at any time a log record is applied to the LOB table space that results in a LOB being marked invalid, the LOB table space is set to auxiliary warning status.

For information about resetting any of these statuses, see Appendix C, “Resetting an advisory or restrictive status” on page 527.

Performing a point-in-time recovery

A recovery operation done with the options TOLOGPOINT, TORBA or TOCOPY is known as a point-in-time recovery. A consistent point-in-time is a quiesce point or an image copy set that was taken with SHRLEVEL REFERENCE. It is not necessary 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 248. DB2 records the RBAs or LRSNs 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, all indexes must be rebuilt using REBUILD INDEX. For more information, see “Resetting the REBUILD pending status” on page 218.

An index cannot be recovered to a prior point-in-time if the index has had its key length increased less than or equal to 16 distinct times since the specified point-in-time. An ALTER that increases the length of the index key column becomes distinct when the unit of work is committed— the ALTERs must take place in 16 different commit scopes. For example, if you ALTER, commit, and then ALTER again, this counts at two distinct ALTERs. Alternatively, if you ALTER, ALTER again and then commit, this counts as one distinct ALTER.

If you use TOLOGPOINT, TORBA or TOCOPY 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 TORBA and TOLOGPOINT options reset the CHECK pending status on table spaces when:

- All members of a table space set are recovered to the same quiesce point or SHRLEVEL REFERENCE point, and no referential constraints were defined on a dependent table after that point. The CHECK pending status is reset for any table space in the table space set.

The TORBA and TOLOGPOINT options set the CHECK pending status on table spaces when:

- One or more members of a table space set are recovered to a previous point in time that is not a common quiesce or SHRLEVEL(REFERENCE) point. Dependent table spaces are placed in CHECK pending status.
- All members of a table space set are recovered to the same quiesce point, but referential constraints were defined on a dependent table after that quiesce point. Table spaces containing those dependent tables are placed in check pending status.
- Table spaces with LOB columns defined were recovered without recovering their LOB table spaces.

The TORBA and TOLOGPOINT options reset the CHECK pending status on indexes when:

- The indexes have been recovered along with the related table space to the same quiesce point or SHRLEVEL REFERENCE point. RECOVER processing resets the CHECK pending status for any indexes in the RECOVER statement.

The TORBA and TOLOGPOINT options set the CHECK pending status on indexes when:

- One or more of the indexes have been recovered to a previous point in time, but the related table space was not recovered in the same RECOVER statement.
- One or more of the indexes have been recovered along with the table space to a previous point in time that is not a quiesce point or SHRLEVEL REFERENCE point.

The auxiliary check pending status (ACHKP) is set on when the CHECK DATA utility detects an inconsistency between a table space with LOB columns defined and a LOB table space. For information about how to reset the ACHKP status, see Appendix C, “Resetting an advisory or restrictive status” on page 527.

For more information about recovering data to a prior point of consistency, see Section 4 (Volume 1) of *DB2 Administration Guide*.

Recovery considerations after rebalancing partitions with REORG: Image copies 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 where 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, DB2 sets restrictive statuses on the 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 nonpartitioning indexes.

For information about resetting these restrictive statuses, see “REORG pending status” on page 532 and “REBUILD pending status” on page 530.

To create a new recovery point, it is strongly recommended that immediately following an ALTER INDEX operation, you either:

- Run REORG with COPYDDN and SHRLEVEL NONE specified, or
- Take a full image copy immediately after REORG completes.

Use RECOVER LOGONLY after data has been redistributed among partitions using REORG. If you perform a point-in-time recovery, you must keep the off-line copies synchronized with the SYSIBM.SYSCOPY records. Therefore, do *not* delete any SYSCOPY='A' records, as they might be needed during the recovery. Only delete

these SYSCOPY records when you are sure you will no longer use the off-line copies taken before the rebalancing REORG.

Planning for point-in-time recovery: TOCOPY, TOLOGPOINT, and TORBA 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 recovery. Copies 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 table space set, and then quiesce them using the QUIESCE utility. This allows RECOVER TORBA to recover the table spaces to the quiesce point with minimal use of the log.

Authorization: Restrict use of TOCOPY, TOLOGPOINT, and TORBA to personnel with a thorough knowledge of the DB2 recovery environment.

Ensuring consistency: RECOVER TORBA, RECOVER TOLOGPOINT, and RECOVER TOCOPY can be used on a single:

- 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, all data sets must be restored to the same level or the data will be inconsistent.

If possible, a table space and all of its indexes (or a table space set and all related indexes) should be recovered in the same RECOVER utility statement, specifying 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, we recommend using the following procedure:

1. RECOVER table spaces to the log point.
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 eliminates the need to run the CHECK INDEX utility.

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 involved in a referential constraint, you should recover all the table spaces involved in a constraint. This is the *table space set.* To avoid setting CHECK pending, you must do both of the following:

- Recover the table space or the table space set to a quiesce point or to an image copy made with SHRLEVEL REFERENCE.

  If you do not recover each table space of the table space set to the same quiesce point, and if any of the table spaces are part of a referential integrity structure:

  — 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 above 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, then the CHECK pending status is set for the table space containing the table with the referential constraint.

For information about actions to take if CHECK INDEX identifies inconsistencies after you perform a RECOVER job, see “Reviewing CHECK INDEX output” on page 75.

For information about resetting the CHECK pending status of table spaces, see “Chapter 2-4. CHECK DATA” on page 55. For information about resetting the CHECK pending status for indexes, see “CHECK pending status” on page 528.

Compressed data: Use caution when recovering a portion of a table space or partition, say one data set, to a prior point in time. If the data set being recovered has been compressed with a different dictionary, then you can no longer read the data. The details of data compression are described in Section 2 (Volume 1) of *DB2 Administration Guide*.

Avoiding specific image copy data sets

You can 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 will always attempt to allocate the data set. The following sections describe the options available if you want to skip a specific image copy data set.

Image copy on tape: If the image copy is on tape, message IEF233D and IEF455D will request the tape for RECOVER.

```
 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 will respond with message DSNU030I and DSNU508I.

```
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 "request denied by operator."

Image copy on DASD: If the image copy is on DASD, you can delete or rename the image copy data set before RECOVER starts executing. RECOVER issues message DSNU030I and DSNU508I.

```
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 "ICF catalog entry not found".

Improving performance

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, then RECOVER uses the log instead.

Include a table space list in your RECOVER utility statement to avoid scanning the log more than once.

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 DASD to a list of objects in parallel.

Optimizing the LOGAPPLY phase

The time required to recover a table space depends also on the time required to read and apply log data. There are several things you can do to optimize the process.

If possible, the required log records are read from the *active* log. That provides the best performance.

Any log records 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 used for archive log data sets is a significant factor in the performance.

- RECOVER a list of objects in one utility statement to take only a single pass of the log.
- Keeping archive logs on DASD provides the best possible performance.
- Controlling archive log data sets by DFSMShsm is next best. DB2 optimizes recall of the data sets. After being recalled, the data set is read from DASD.
- 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 below.

Which log data sets to use and where they reside is recorded in the BSDS, which must be kept current. If the archive log data sets are cataloged, the integrated catalog facility catalog tells where to allocate the required data set.

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 Section 2 of *DB2 Installation Guide*.

DFSMShsm data sets: Recall for the first DFSMShsm archive log data set starts automatically when the LOGAPPLY phase starts. When recall is complete and the first log record is read, recall for the next archive log data set starts. This process is known as *look-ahead* recalling. Its object is to recall the next data set in parallel with reading 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 *not* marked with an asterisk (*) is *required* for the job to complete. These volumes should be obtained 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 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

```
-SET ARCHIVE COUNT (10)
```

to assign 10 tape units to your DB2 subsystem.

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.

The amount of time DB2 delays deallocation can be dynamically changed with the TIME option of the SET ARCHIVE command. For example, use:

```
-SET ARCHIVE TIME(60)
```

to specify a 60 minute delay. In a data sharing environment, you might want to specify (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 DASD.

2. Consider staging cataloged tape data sets to DASD before allocation by the log read process.
3. If the data sets are read from tape, set both the COUNT and the TIME value to the maximum allowable within the system constraints.

Recovering image copies in a JES3 environment

Ensure that there are sufficient units available to mount the required image copies. In a JES3 environment, if the number of image copies 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. The status can be turned off in several ways, listed below:

- 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.
- Run REORG INDEX SORTDATA on the affected index.

Considerations for running RECOVER

This section includes additional information to keep in mind when running RECOVER.

Allocating incremental image copies

RECOVER will attempt to dynamically allocate ALL required incremental image copy data sets. If any of the incremental image copies are missing, RECOVER will:

- Identify the first incremental image copy that is missing
- Use the incremental image copies up to the missing incremental image copy
- Not use the remaining incremental image copy data sets
- Apply 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 not enough tape drives are available, RECOVER will NOT use the remaining incremental image copy data sets.

Performing fallback recovery

If the RECOVER utility cannot use the latest primary copied data set as a starting point for recovery, it attempts to use the backup copied data set, if one is available. If neither image copy is usable, it attempts to fall back to a previous recoverable point. If a previous REORG LOG YES or LOAD REPLACE LOG YES was done, it attempts to recover from the log. If there are no good full image copies, and no

previous REORG LOG YES or LOAD REPLACE LOG YES, the RECOVER utility terminates.

If RECOVER processes an index for which no full copy exists, or if the copy cannot be used due to utility activity that occurred on the index or on its underlying table space, the index remains untouched and utility processing terminates with return code 8. For more information about this situation, see "Setting and clearing the informational COPY pending status" on page 103.

If you always make multiple image copies, RECOVER should seldom fall back to an earlier point. Instead, RECOVER relies on the backup copied data set should the primary copied data set be unusable.

In a JES3 environment, fallback recovery can be accomplished by issuing a JES3 "cancel,s" command at the time the allocation mount message is issued. This might be necessary in the case where a volume is not available or the given volume is not desired.

Retaining tape mounts

If the image copy data sets from which you want to recover reside on the same tape, you do not need to remove the tape. For noncataloged image copies, specify the following parameters on the DD cards (in this example, the DD cards are *ddname1* and *ddname2*):

```
//ddname1   DD   UNIT=3480,DSN=data-set-name1,DISP=(OLD,PASS),LABEL=1,
//               VOL=(,RETAIN,SER=vol-ser)
//ddname2   DD   UNIT=3480,DSN=data-set-name2,DISP=(OLD,PASS),LABEL=2,
//               VOL=(,REF=*.ddname1)
```

This example only works for multiple image copies on a single volume. To use multiple image copies on multiple volumes, the image copy data sets must be cataloged. For cataloged image copies on one or more tape volumes, specify the following parameters on the DD cards (in this example, the DD cards are *ddname1*, *ddname2*, and *ddname3*):

```
//ddname1   DD   DSN=data-set-name1,UNIT=3480,DISP=(OLD,PASS),VOL=(,RETAIN),
//               LABEL=(1,SL)
//ddname2   DD   DSN=data-set-name2,UNIT=3480,DISP=(OLD,PASS),VOL=(,RETAIN),
//               LABEL=(2,SL)
//ddname3   DD   DSN=data-set-name3,UNIT=3480,DISP=(OLD,PASS),VOL=(,RETAIN),
//               LABEL=(3,SL)
```

Avoiding damaged media

When a media error is detected, DB2 prints a message giving 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 can re-access the damaged media. You must:

1. Use -ALTER STOGROUP to remove the bad volume and add another
2. Execute the RECOVER utility for all objects on that volume.

If the RECOVER utility cannot complete because of severe errors caused by the damaged media, it can be necessary to use access method services (IDCAMS) to delete the cluster for the table space or index with the NOSCRATCH option. Refer to the access method services reference manual for details. If the table space or index is defined 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

For instructions on restarting a utility job, see “Restarting an online utility” on page 48.

Terminating RECOVER: Terminating a RECOVER job with the -TERM UTILITY command leaves the table space being recovered in RECOVER pending status, and the index space being recovered in the REBUILD pending status. If you are recovering a table space to a previous point-in-time, then its indexes are left in the REBUILD pending status. The data is unavailable until the object has been successfully recovered or rebuilt.

Restarting RECOVER: RECOVER can be restarted at the beginning of the phase or at the current checkpoint.

If you attempt to recover multiple objects 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 or REBUILD pending status. The status of the remaining objects is unchanged.
- The LOGAPPLY phase: All objects specified in the RECOVER statement are placed in the RECOVER or REBUILD pending status.

In both cases, you must identify and fix the causes of the failure before a current restart is performed.

If you specified the PARALLEL keyword in your RECOVER utility statement, use RESTART(PHASE) to restart at the beginning of the current phase, or restart from the last commit point of each object processed in parallel.

Concurrency and compatibility

Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions of the same table space or index space are compatible. However, if a nonpartitioning index exists on a partitioned table space, utilities operating on different partitions of a table space can be incompatible because of contention on the nonpartitioning index.

Table 43 on page 251 shows which claim classes RECOVER claims and drains and any restrictive state the utility sets on the target object.

Table 44 on page 251 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 is also shown.

Table 43. Claim classes of RECOVER operations. Use of claims and drains; restrictive states set on the target object.

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]
Index or physical partition	DA/UTUT	DA/UTUT	DA/UTUT	DA/UTUT CW/UTRW[1]
Nonpartitioning index	DA/UTUT	DA/UTUT	DR	DA/UTUT CW/UTRW[1]
RI dependents		CHKP (YES)	CHKP (YES)	

Legend:

- CHKP (YES): Concurrently running applications see 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
- UTRW: Utility restrictive state, read/write access allowed
- UTUT: Utility restrictive state, exclusive control
- Blank: Object is not affected by this utility

Notes:

1. During the UTILINIT phase, the claim and restrictive state change from DA/UTUT to CW/UTRW.

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

Table 44 (Page 1 of 2). RECOVER compatibility

Action	RECOVER (no option)	RECOVER TOCOPY or TORBA	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

Table 44 (Page 2 of 2). RECOVER compatibility

Action	RECOVER (no option)	RECOVER TOCOPY or TORBA	RECOVER ERROR-RANGE
QUIESCE	No	No	No
REBUILD INDEX	No	No	No
REORG INDEX	Yes	No	Yes
REORG TABLESPACE	No	No	No
REPAIR LOCATE TABLESPACE	No	No	No
REPAIR LOCATE INDEX	Yes	No	Yes
REPORT	Yes	Yes	Yes
RUNSTATS INDEX	No	No	No
RUNSTATS TABLESPACE	No	No	No
STOSPACE	Yes	Yes	Yes

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.

Sample control statements

Example 1: Recover an error range. Recover from reported media failure in partition 2 of table space DSN8D61A.DSN8S61D.

```
//STEP5    EXEC DSNUPROC,UID='HUIAU326.RESTORE',TIME=1440,
//         UTPROC='',
//         SYSTEM='V61A',DB2LEV=DB2A
//SYSIN    DD *
RECOVER TABLESPACE DSN8D61A.DSN8S61D DSNUM 2 ERROR RANGE
/*
```

Example 2: Recover a table space. Recover table space DSN8S61D, in database DSN8D61A.

```
RECOVER TABLESPACE DSN8D61A.DSN8S61D
```

Example 3: Recover a table space partition. Recover the second partition of table space DSN8S61D.

```
RECOVER TABLESPACE DSN8D61A.DSN8S61D DSNUM 2
```

Example 4: Recover a table space to a specific RBA. Recover table spaces DSN8D61A.DSN8S61E and DSN8D61A.DSN8S61D to their quiesce point (RBA `X'000007425468'`).

```
RECOVER TABLESPACE DSN8D61A.DSN8S61E DSNUM 2
        TABLESPACE DSN8D61A.DSN8S61D
        TORBA X'000007425468'
```

Example 5: Recover a list of objects to a point in time. The point in time is the common LRSN value from the SYSIBM.SYSCOPY records for the list of objects in the COPY SHRLEVEL REFERENCE job on page 109. The objects in the list are synchronized after successful completion of this RECOVER utility statement. This example restores four objects in parallel.

```
RECOVER TOLOGPOINT X'1234567890AB' PARALLEL(4)
      TABLESPACE DSN8D61A.DSN8S61D
      INDEX DSN8610.XDEPT1
      INDEX DSN8610.XDEPT2
      INDEX DSN8610.XDEPT3
      TABLESPACE DSN8D61A.DSN8S61E
      INDEX DSN8610.XEMP1
      INDEX DSN8610.XEMP2
```

Chapter 2-15. REORG INDEX

The 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 collect inline statistics 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 will produce a report detailing if a REORG is recommended; a REORG is not performed.

For a diagram of REORG INDEX syntax and a description of available options, see "Syntax and options of the control statement" on page 256. For detailed guidance on running this utility, see "Instructions for running REORG INDEX" on page 264.

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 the partitioning index.

Authorization required: To execute this utility on a user index, the privilege set of the process must include one of the following:

- 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, the privilege set of the process must include one of the following:

- 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

An authority other than installation SYSADM or installation SYSOPR can receive message DSNT500I, "resource unavailable," while trying to reorganize an index space in the catalog or directory. This can happen 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, the privilege set must include the SELECT privilege on the catalog tables.

Execution phases of REORG INDEX: The REORG utility operates in these phases:

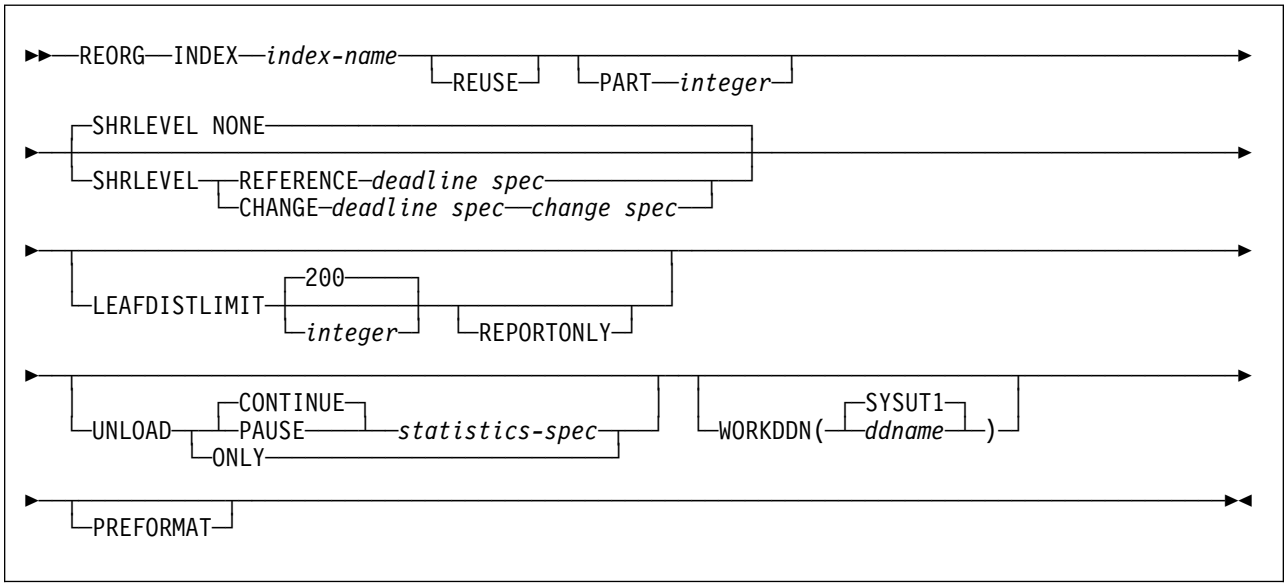
Phase	Description
UTILINIT	Initialization and setup
UNLOAD	Unloads index space and writes keys to a sequential data set.
BUILD	Builds indexes. Updates index statistics.
LOG	Processes log iteratively. Used only if you specify SHRLEVEL CHANGE.
SWITCH	Switches access to shadow copy of index space or partition. Used only if you specify SHRLEVEL REFERENCE or CHANGE.
UTILTERM	Cleanup

Syntax and options of the control statement

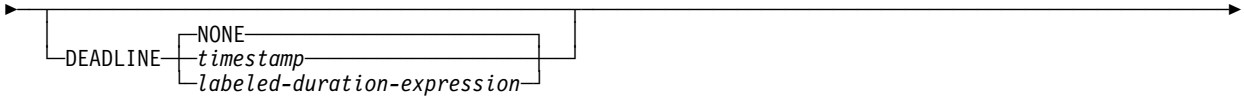
The utility control statement defines the function the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name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

Syntax diagram

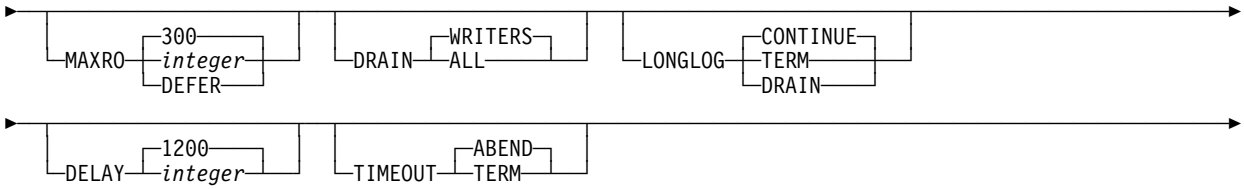
For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.



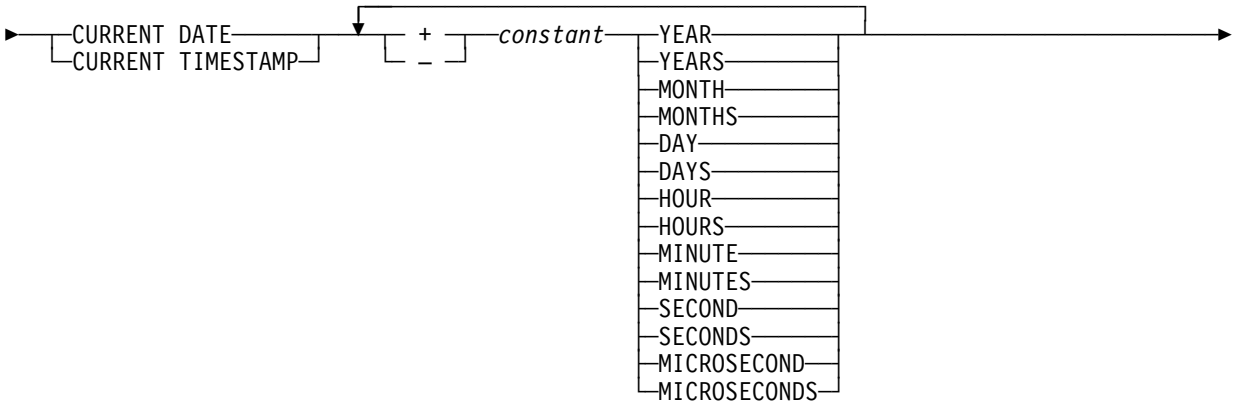
deadline spec:



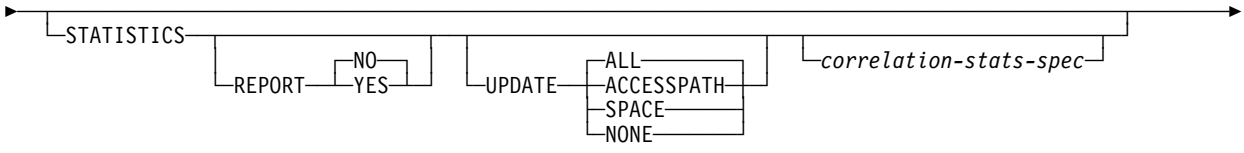
change spec:



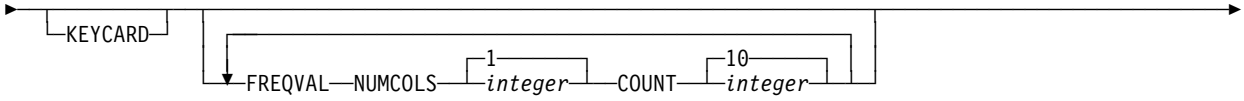
labeled-duration-expression:



statistics-spec:



correlation-stats-spec:



Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see “Control statement coding rules” on page 27.

INDEX *index-name*

Specifies an index to be reorganized.

index-name is the qualified name of the index, in the form *creator-id.index-name*. If you omit the qualifier creator ID, the user identifier for the utility job is used.

REUSE When used with SHRLEVEL NONE, specifies that REORG logically resets and reuses 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 will not be released if you use the REUSE parameter.

REUSE does not apply if you also specified SHRLEVEL REFERENCE or CHANGE.

PART *integer* Identifies a partition to be reorganized. You can reorganize a single partition of a partitioning index. *integer* must be in the range from 1 to the number of partitions that are defined for the partitioning index. The maximum is 254.

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 allowed during the RELOAD phase of REORG.

NONE Specifies that reorganization operates by unloading from the area 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), the following parameters cannot be specified: MAXRO, LONGLOG, DELAY, and DEADLINE.

REFERENCE Specifies that reorganization operates by unloading from the area being reorganized (while applications can read but cannot write to the area), building into a shadow copy of that area (while applications can read but cannot write to the original copy), switching applications' future access 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 used by REORG INDEX" on page 266.

If you specify REFERENCE, you cannot specify the following parameters:

- UNLOAD. Reorganization with REFERENCE always performs UNLOAD CONTINUE.
- MAXRO, LONGLOG, and DELAY.

CHANGE Specifies that reorganization operates by unloading from the area being reorganized (while applications can read and write to the area), building 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 applications' future access 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 used by REORG INDEX" on page 266.

If you specify CHANGE, you cannot specify the following parameters:

- UNLOAD. Reorganization with CHANGE always performs UNLOAD CONTINUE.

DEADLINE Specifies the deadline for the switch phase to finish. If DB2 estimates that the SWITCH phase will not finish by the deadline, DB2 issues the messages that the -DISPLAY UTILITY command would issue and then terminates reorganization.

NONE Specifies that there is no deadline by which the SWITCH phase of log processing must finish.

The **default** is **NONE.**

timestamp
timestamp specifies the deadline for the SWITCH phase of log processing to finish. 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 finish. The calculation is either based on CURRENT TIMESTAMP or CURRENT DATE. This deadline must not have already occurred when REORG is executed.

For example, to ensure that the SWITCH phase is complete by 6:30 AM two days from now, use the following expression:

```
DEADLINE CURRENT DATE + 2 DAYS + 6 HOURS + 30 MINUTES
```

If REORG SHRLEVEL REFERENCE or SHRLEVEL CHANGE terminates due to a DEADLINE specification, DB2 issues message DSNU374I with reason code 2, but does not set a restrictive status.

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 value specified with MAXRO.

The ALTER UTILITY command can change the value of MAXRO.

The **default** is **300** seconds. The value must be an integer.

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 will occur. The message also suggests that this would be an appropriate time to execute ALTER UTILITY to change the MAXRO value and thus cause REORG to finish. 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 just 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 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:

- There is a lot of SQL update activity during the log phase.
- The default behavior results in a large number of -911 SQL error messages.

LONGLOG Specifies the action that DB2 performs, after sending a message to the console, if the number of records that the next iteration of log process will process is not sufficiently lower than the number that the previous iterations processed. This situation means that reorganization's reading of the log is not catching up to applications' writing of the log quickly enough.

CONTINUE Specifies that until the time on the JOB statement expires, DB2 continues performing reorganization, including iterations of log processing, if the estimated time to perform an iteration exceeds the time specified with MAXRO.

A value of DEFER for MAXRO and a value of CONTINUE for LONGLOG together mean that REORG continues allowing access to the original copy of the area 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 will terminate reorganization after the delay specified by the DELAY parameter.

DRAIN Specifies that DB2 drains the write claim class after the delay 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 specified by the LONGLOG parameter.

integer is the number of seconds. The value must be an integer. The **default** is **1200**.

TIMEOUT Specifies the action 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 If a time out condition occurs, DB2 leaves the objects in a UTRO or UTUT state.

TERM If you specify the TERM option and a time out condition occurs, then DB2:

1. Issues an implicit TERM UTILITY command, causing the utility to end with a return code 8
2. Issues the DSNU590I and DSNU170I messages
3. Leaves the objects in a RW state.

LEAFDISTLIMIT

The specified value is compared to LEAFDIST for the specified partitions in SYSIBM.SYSINDEXPART for the specified index. If any LEAFDIST exceeds the value specified for LEAFDISTLIMIT, REORG is performed or recommended.

The **default** value is **200**.

REPORTONLY

If you specify this option, the REORG is only 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 end after the data is unloaded.

CONTINUE Specifies that, after the data has been unloaded, the utility continues processing.

The **default** is **CONTINUE**.

PAUSE Specifies that after the data has been unloaded, processing ends. 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).

ONLY Specifies that after the data has been unloaded, the utility job ends and the status in SYSIBM.SYSUTIL corresponding to this utility ID is removed.

STATISTICS Specifies the gathering of statistics for the index; the statistics are either reported or stored in the DB2 catalog.

REPORT Determines if a set of messages is generated to report the collected statistics.

NO Indicates that the set of messages is not output to SYSPRINT.

The **default** is **REPORT NO**.

YES Indicates that the set of messages is output to SYSPRINT. The messages generated are dependent on the combination of keywords (such as TABLESPACE, INDEX, TABLE, and COLUMN) 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 Tells whether the collected statistics are inserted into the catalog tables. UPDATE also allows you to select statistics used for access path selection or statistics used by database administrators.

ALL Indicates that all collected statistics will be updated in the catalog.

The **default** is **UPDATE ALL**.

ACCESSPATH Indicates that only the catalog table columns that provide statistics used for access path selection are updated.

SPACE Indicates that only the catalog table columns that provide statistics to help the database administrator assess the status of a particular table space or index are updated.

NONE Indicates that no catalog tables are updated with the collected statistics. This option is only valid when REPORT YES is specified.

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.

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 to concatenate together when collecting frequent values from the specified index. Specifying '3' means to collect frequent values on the concatenation of the first three key columns. The default is 1, which means collect frequent values on the first key column of the index.

COUNT Indicates the number of frequent values to be collected. Specifying '15' means collect 15 frequent values from the specified key columns. The default is *10*.

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**.

Even though WORKDDN is an optional keyword, a DD card for the unload output data set is required in the JCL. If you do not specify WORKDDN, or if you specify it without a *ddname*, the JCL must have a DD card with the name SYSUT1. If *ddname* is given, then a DD card must be supplied that matches it.

PREFORMAT Specifies that the remaining pages are 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 on PREFORMAT, see “Improving performance with LOAD or REORG PREFORMAT” on page 162.

Instructions for running REORG INDEX

To run REORG INDEX, you must:

1. Read “Before running REORG INDEX” in this chapter.
2. Prepare the necessary data sets, as described in “Data sets used by REORG INDEX” on page 266.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27. (For examples of JCL for REORG INDEX, see “Sample control statements” on page 275.)
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 267. (For a complete description of the syntax and options for REORG INDEX, see “Syntax and options of the control statement” on page 256.)
5. Check the compatibility table in “Concurrency and compatibility” on page 273 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 271.
7. Run REORG.

See “Chapter 2-1. Invoking DB2 online utilities” on page 27 for an explanation of ways to execute DB2 utilities.

Before running REORG INDEX

Region size: The recommended minimum region size is 4096K.

User-managed data sets and SHRLEVEL REFERENCE and CHANGE: If an index or partition to be reorganized resides in user-managed data sets, then before executing the REORG utility with SHRLEVEL REFERENCE or SHRLEVEL CHANGE, you must create shadow data sets with the names and attributes described in Section 2 (Volume 1) of *DB2 Administration Guide*. The data sets must already exist when you execute REORG.

The names have the form `catname.DSNDBx.dbname.psname.S0001.Annn`. Define the data sets as LINEAR and use SHAREOPTIONS(3,3). An efficient method for defining shadow data sets specifies MODEL, so the shadow is created like the original. For example:

```
DEFINE CLUSTER +
       (NAME('catname.DSNDBC.dbname.psname.S0001.A001') +
       MODEL('catname.DSNDBC.dbname.psname.I0001.A001')) +
      DATA                                               +
       (NAME('catname.DSNDBD.dbname.psname.S0001.A001') +
       MODEL('catname.DSNDBD.dbname.psname.I0001.A001') )
```

If an index, or partition 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. You can create the shadows ahead of time for DB2-managed data sets, and it is strongly recommended that you do so for the shadow data set of the logical partition of nonpartitioning indexes.

Regardless of whether the area being reorganized resides in user-managed or DB2-managed data sets, data sets with names that have the form `catname.DSNDBx.dbname.psname.T0001.Annn` must not already exist when you execute REORG with SHRLEVEL REFERENCE or SHRLEVEL CHANGE.

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

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 held.

Data sharing considerations for REORG: You must 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.

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.

There is one RECOVER pending restrictive state:

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 not in RECP. RECP can be reset by recovering only the single logical partition.

There are three REBUILD pending restrictive states:

RBDP REBUILD pending status (RBDP) is set on a physical or logical index partition. The individual physical or logical partition is inaccessible and must be rebuilt using the REBUILD INDEX utility.

PSRBD Page set REBUILD pending (PSRBD) is set for nonpartitioning indexes. The entire index space is inaccessible and must be rebuilt 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 is made available again when the affected partitions are rebuilt using the REBUILD INDEX utility.

For information about resetting the REBUILD pending and RECOVER pending states, see Table 93 on page 531 and Table 92 on page 530.

CHECK pending status: You cannot reorganize an index when the data is in the CHECK pending status. See "Chapter 2-4. CHECK DATA" on page 55 for more information about resetting the CHECK pending status.

Data sets used by REORG INDEX

Table 53 on page 307 describes the data sets used by REORG. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 45. Data sets used by REORG INDEX

Data Set	Description	Required?
SYSIN	Input data set containing the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
Work data set	One temporary data set for unload output and build input. The DD name is specified with the WORKDDN option of the utility control statement. The default DD name for this data set is SYSUT1.	Yes

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

Index The name of the index space to be reorganized. It is named in the control statement and is accessed through the DB2 catalog.

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 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.

To calculate the approximate size (in bytes) of the WORKDDN data set SYSUT1, follow these steps:

1. Calculate the number of keys:

```
number of keys = #tablerows
```

where:

#tablerows Number of records in the table.

2. Multiply the number of keys by the key length plus 9.

Creating the control statement

See “Syntax and options of the control statement” on page 256 for REORG syntax and option descriptions. See “Sample control statements” on page 275 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 object should be reorganized”
“Determining when an index requires reorganization”
“Specifying access with SHRLEVEL” on page 268
“Unloading without reloading” on page 270
“Considerations for fallback recovery” on page 270
“Changing data set definitions” on page 270
“Temporarily interrupting REORG” on page 270
“Improving performance” on page 270
“Improving performance with LOAD or REORG PREFORMAT” on page 162

Product-sensitive Programming Interface

Determining when an object should be reorganized

You can determine when to run REORG for indexes by using the LEAFDISTLIMIT catalog query option. If you specify the REPORTONLY option, REORG will produce a report detailing if 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 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

Determining when an index requires reorganization

Use this query to identify user created indexes and DB2 catalog indexes that you should consider reorganizing with the REORG utility:

Product-sensitive Programming Interface

```
SELECT IXNAME, IXCREATOR
 FROM SYSIBM.SYSINDEXPART
   WHERE LEAFDIST > 200;
```

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 rule of thumb for general cases. It is not absolute. There are cases where 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 higher than 200 can be acceptable.

After you run RUNSTATS, the following SQL statement provides the average distance (multiplied by 100) between successive leaf pages during sequential access of the ZZZ index.

Product-sensitive Programming Interface

```
SELECT LEAFDIST
  FROM SYSIBM.SYSINDEXPART
  WHERE IXCREATOR = 'index_creator_name'
  AND IXNAME = 'index_name';
```

End of Product-sensitive Programming Interface

If LEAFDIST increases over time, this probably indicates that the index should 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, due to empty pages for FREEPAGE and non-leaf pages.

Specifying access with SHRLEVEL

For reorganizing an index, or a partition of a index, the SHRLEVEL option lets you choose the level of access you have to your data during reorganization:

- REORG with SHRLEVEL NONE, the default, reloads the reorganized data into the original area 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 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 being reorganized. Applications can read and write 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 specified by MAXRO. If this condition is met, the next iteration will be 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 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 specified by DELAY and then performs the action specified by LONGLOG.

Operator actions: LONGLOG specifies the action that DB2 performs if log processing is not catching up. See “Option descriptions” on page 258 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 you may take:

- 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 increases the likelihood that log processing will catch up. 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 allocated to reorganization and to applications. This can increase the likelihood that log processing will catch up. 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 can increase the likelihood that log processing will catch up. 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 will be less than or equal to the time specified in the MAXRO phrase.
- REORG terminates for any reason (including the deadline).

Unloading without reloading

REORG can unload data without continuing and without leaving a SYSIBM.SYSUTIL record after the job ends.

Considerations for fallback recovery

Successful REORG INDEX processing inserts an SYSIBM.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. The user effectively changes the characteristics of a user-managed data set by specifying the desired new characteristics when creating the shadow data set; see page 264 for more information about user-managed data sets. In particular, placing the original and shadow data sets on different DASD 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 using the phase restart or current restart. The REORG statement must not be altered.

The SYSIBM.SYSUTIL record for the REORG utility remains in "stopped" status until REORG is restarted or terminated.

While REORG is interrupted by PAUSE, you may 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 it takes to run REORG INDEX on partitions of a partitioned index is roughly the same as it would take to run a single REORG index job. The elapsed time is a fraction of what it would take to run 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.

Terminating or restarting REORG INDEX

If you terminate REORG with the TERM UTILITY command during the UNLOAD phase, objects have not yet been changed and the job can be rerun.

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 there is a problem 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 abends or is terminated, the objects are left in RECOVER pending status. See Appendix C, “Resetting an advisory or restrictive status” on page 527 for information about resetting either status.

Table 46. REORG INDEX phases and restrictive statuses

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 or objects may 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, sets the index in RBDP status at the beginning of the phase, and resets RBDP at the end of the phase.

Restarting REORG: Table 47 on page 272 provides information about restarting REORG INDEX.

If you restart REORG in the UTILINIT phase, it re-executes from the beginning of the phase. If REORG abends or 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 allowed (CURRENT and PHASE). "None" indicates that no restart is allowed. A blank indicates that the phase does not occur. The "Data Sets Required" column lists the data sets that must exist to perform the specified type of restart in the specified phase.

Table 47. REORG INDEX utility restart information

Phase	Type for NONE	Type for REFERENCE	Type for CHANGE	Data Sets Required	Notes
UNLOAD	CURRENT PHASE	CURRENT PHASE	None None	SYSUT1	
BUILD	CURRENT PHASE	CURRENT PHASE	None None	SYSUT1 SYSUT1	1
LOG			None None		
SWITCH		CURRENT PHASE	CURRENT PHASE	originals and shadows 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 using RESTART CURRENT, inline statistics collection will 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 will be updated using RESTART(PHASE), see Table 48.

Table 48. REORG INDEX utility phase restart using STATISTICS keyword

Phase	CURRENT	PHASE
UTILINIT	NO	YES
UNLOAD	NO	YES
BUILD	NO	YES

For instructions on restarting a utility job, see “Chapter 2-1. Invoking DB2 online utilities” on page 27.

Restarting REORG after an out of space condition: See “Restarting after the output data set is full” on page 49 for guidance in restarting REORG from the last commit point after receiving an out of space condition.

Concurrency and compatibility

Individual index partitions are treated as distinct target objects. Utilities operating on different partitions of the same index space are compatible.

REORG INDEX compatibility

Table 49 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 50 on page 274 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.

Table 49. Claim classes of REORG INDEX operations. Use of claims and drains; restrictive states set on the target object.

Phase	REORG INDEX SHRLEVEL NONE	REORG INDEX SHRLEVEL REFERENCE	REORG INDEX SHRLEVEL CHANGE
UNLOAD	DW/UTRO	DW/UTRO	CR/UTRW
BUILD	DA/UTUT		
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
- Blank: Any claim, drain, or restrictive state for this object does not change in this phase.

Notes:

1. Applicable if you specified DRAIN ALL.

REORG INDEX does not set a utility restrictive state if the target object is an index on DSNDB01.SYSUTILX.

Table 50. REORG INDEX compatibility

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
RECOVER INDEXSPACE	No
RECOVER TABLESPACE (no option)	Yes
RECOVER TABLESPACE TOCOPY or TORBA	No
RECOVER TABLESPACE ERROR RANGE	Yes
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 without cluster index	Yes
REORG TABLESPACE SHRLEVEL NONE UNLOAD ONLY or EXTERNAL with cluster index	No
REPAIR LOCATE KEY	No
REPAIR LOCATE RID DUMP, VERIFY, or REPLACE	Yes
REPAIR LOCATE RID DELETE	No
REPAIR LOCATE TABLESPACE PAGE REPLACE	Yes
REPAIR LOCATE INDEX PAGE REPLACE	No
REPORT	Yes
RUNSTATS INDEX	No
RUNSTATS TABLESPACE	Yes
STOSPACE	Yes

To run on SYSIBM.DSNLUX01 or SYSIBM.DSNLUX02, REORG INDEX must be the only utility in the job step and the only utility running in the DB2 subsystem.

Reviewing REORG INDEX output

The output from REORG INDEX consists of a reorganized index or index partition. Table 51 summarizes the effect of REORG.

Table 51. REORG INDEX summary

Specification	Results
REORG INDEX	Entire index (all parts if partitioned)
REORG INDEX PART *n*	Part *n* of partitioning 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. (Those values can be set by the CREATE INDEX, or ALTER INDEX statements). REORG leaves one free page after reaching the FREEPAGE limit for each table in the index space.

Sample control statements

Example 1: Reorganizing an index. Reorganize index XMSGTXT1. Stop the utility after the index keys have been unloaded, but allow for subsequent restart.

```
REORG INDEX DSN8610.XMSGTXT1
  UNLOAD PAUSE
```

Example 2: REORG INDEX using STATISTICS. Reorganize the index XEMPL1, using the STATISTICS option to update the catalog table statistics for this index.

```
REORG INDEX DSN8610.XEMPL1
   SHRLEVEL REFERENCE STATISTICS
```

Chapter 2-16. REORG TABLESPACE

The REORG TABLESPACE utility reorganizes a table space to improve access performance and 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 collect inline statistics 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 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 will produce a report detailing if a REORG is recommended; a REORG is not performed.

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.

For a diagram of REORG TABLESPACE syntax and a description of available options, see “Syntax and options of the control statement” on page 279. For detailed guidance on running this utility, see “Instructions for running REORG TABLESPACE” on page 303.

Output: If the table space or partition has the COMPRESS YES attribute, then the data is compressed when reloaded. If you specify the KEEPDICTIONARY option of REORG, the current dictionary is used; otherwise a new dictionary is built.

REORG TABLESPACE can be executed on the table spaces in the DB2 catalog database (DSNDB06) and some table spaces in the directory database (DSNDB01). It cannot be executed on any table space in the DSNDB07 database.

Table 52. Summary of REORG TABLESPACE output

REORG Specified	Results
REORG TABLESPACE	Reorganizes all data, entire partitioning index, and all non-partitioning indexes.
REORG TABLESPACE PART *n*	Reorganizes data for PART *n*, PART *n* of the partitioning index, and index entries for PART *n* in all nonpartitioning indexes.
REORG TABLESPACE PART *n:m*	Reorganizes data for PART *n* through PART *m*, parts *n* through *m* of the partitioning index, and index entries for those parts in all nonpartitioning indexes.

Authorization required: To execute this utility on a user table space, the privilege set of the process must include one of the following:

- 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, the privilege set of the process must include one of the following:

- 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 use REORG TABLESPACE SHRLEVEL CHANGE, the privilege set must include DELETE, INSERT, SELECT, and UPDATE privileges on the mapping table (see page 304).

To run REORG TABLESPACE STATISTICS REPORT YES, the privilege set must include the SELECT privilege on the catalog tables.

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 can happen 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.

Execution phases of REORG TABLESPACE: The REORG TABLESPACE utility operates in these phases:

Phase	Description
UTILINIT	Initialization and setup
UNLOAD	Unloads table space; sorts data if a clustering index exists and you specified either SORTDATA or SHRLEVEL CHANGE. If you specified NOSYSREC, passes rows in memory to the RELOAD phase, otherwise writes them to a sequential data set.
RELOAD	Reloads from the sequential data set into the table space; creates full image copies if you specify COPYDDN, RECOVERYDDN, SHRLEVEL REFERENCE, or SHRLEVEL CHANGE. If you specify the SORTKEYS option, a subtask sorts the index keys. Updates table and table space statistics.
SORT	Sorts index keys. If you specify the SORTKEYS option, the sorted keys are passed in memory to the BUILD phase.
BUILD	Builds indexes. Updates index statistics.
SORTBLD	If you specify a parallel index build using the SORTKEYS keyword, all activities that normally occur in both the SORT and BUILD phases occur in the SORTBLD phase instead.
LOG	Processes log iteratively; appends changed pages to the full image copies. Used only if you specify SHRLEVEL CHANGE.
SWITCH	Switches access to shadow copy of table space or partition. Used only if you specify SHRLEVEL REFERENCE or CHANGE.
BUILD2	Corrects nonpartitioning indexes if you specify REORG TABLESPACE PART SHRLEVEL REFERENCE or CHANGE.
UTILTERM	Cleanup

Execution phases of REORG TABLESPACE on a LOB table space: The REORG TABLESPACE utility operates in these three phases when you run it on a LOB table space:

Phase	Description
UTILINIT	Initialization and setup
REORGLOB	Rebuilds the LOB table space in place; no LOBs are unloaded or reloaded. The LOB table space is set to RECOVER pending status at the start of processing; this status is reset on REORGLOB completion. If the REORGLOB phase fails, the LOB table space remains in RECOVER pending status.
UTILTERM	Cleanup

You cannot restart REORG TABLESPACE on a LOB table space in the REORGLOB phase. Before executing REORG TABLESPACE on a LOB table space defined with LOG NO, you should take a full image copy to ensure recoverability.

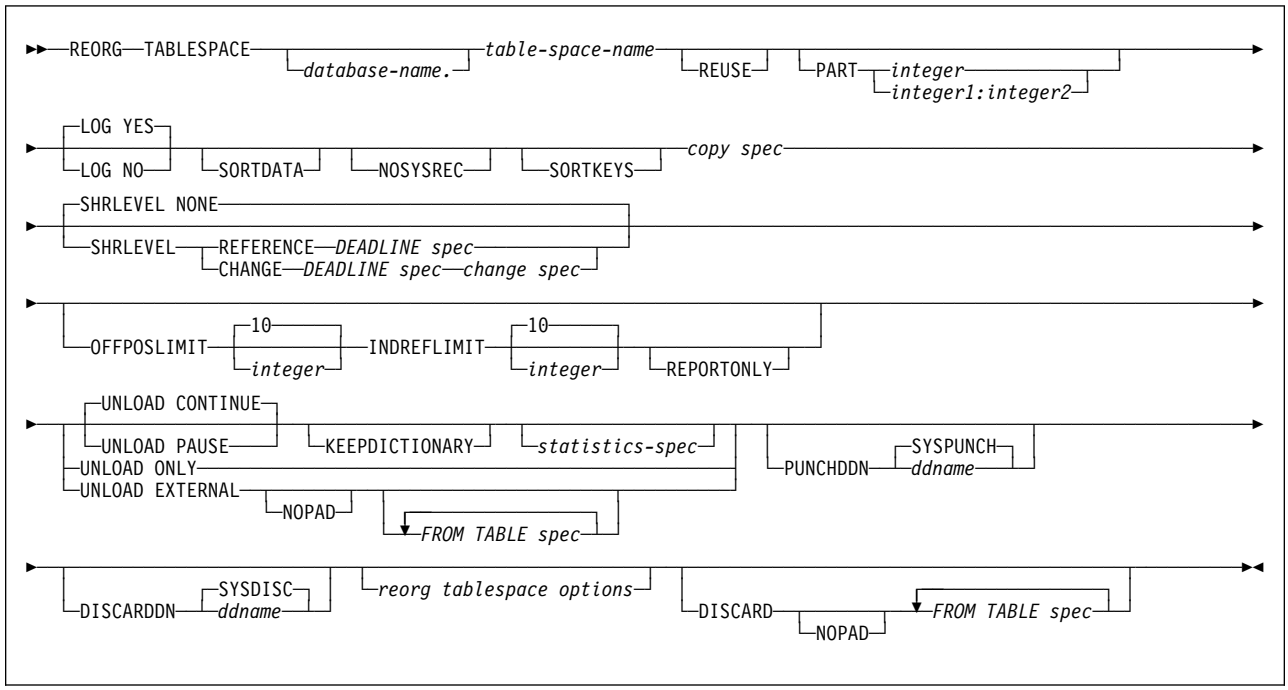
If the LOB table space was defined with LOG NO, it is left in COPY pending status after REORG TABLESPACE completes processing.

Syntax and options of the control statement

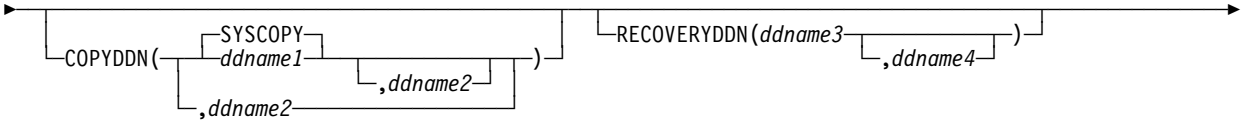
The utility control statement defines the function the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name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

Syntax diagram

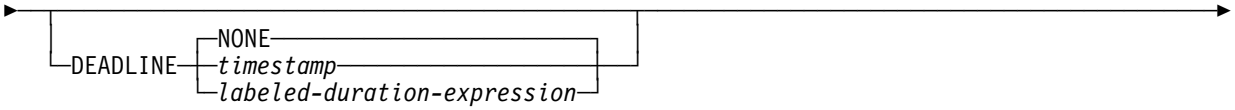
For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.



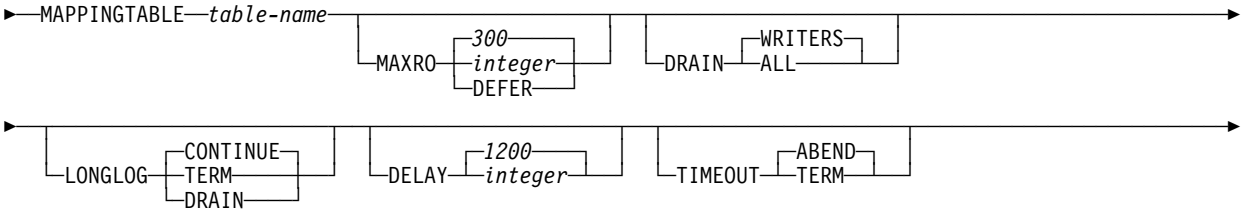
copy spec:



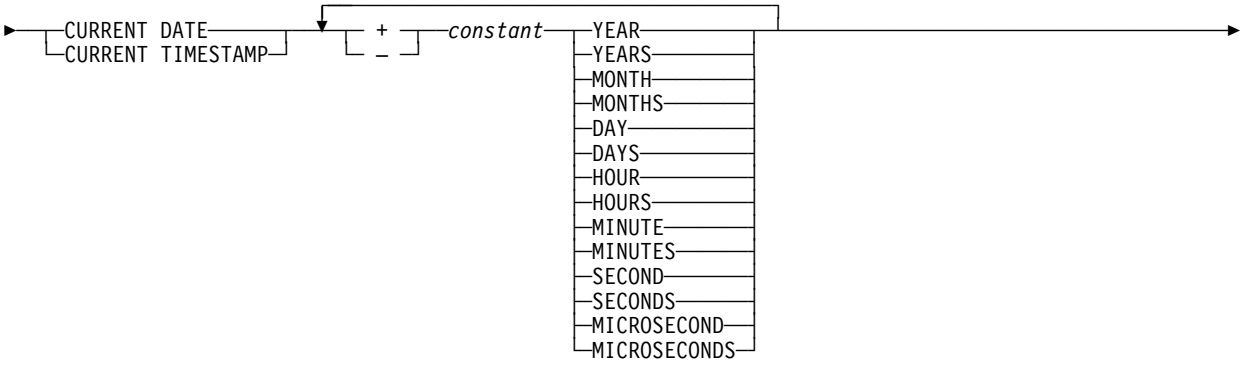
DEADLINE spec:



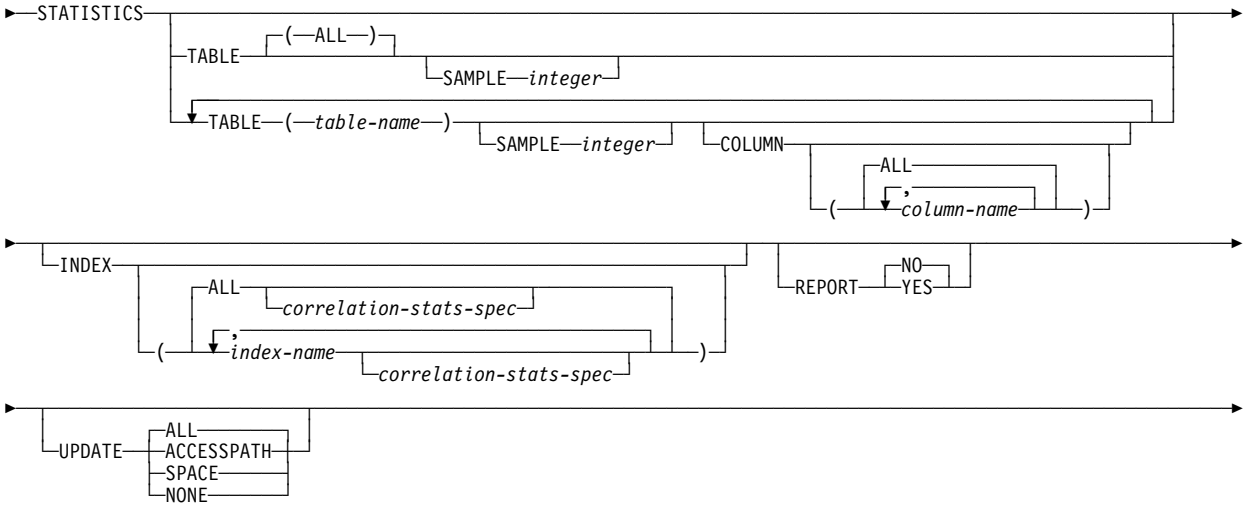
change spec:



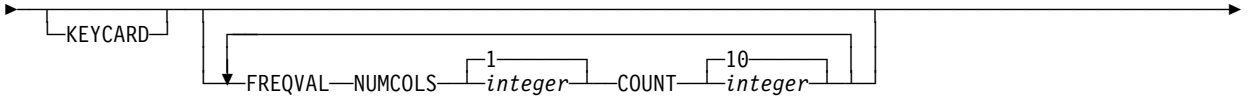
labeled-duration-expression:



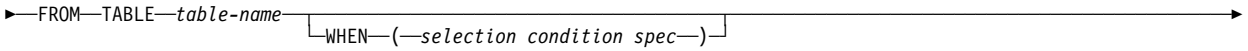
statistics-spec:



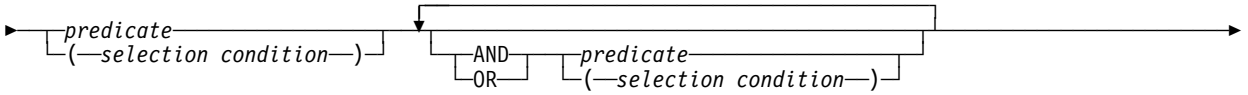
correlation-stats-spec:



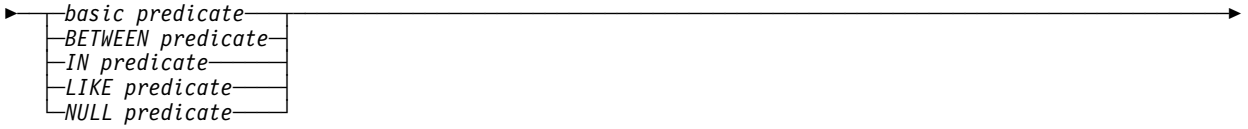
FROM TABLE spec:



selection condition spec:



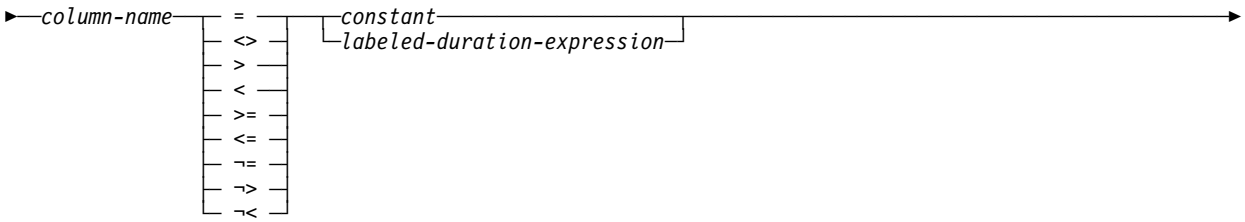
predicate:



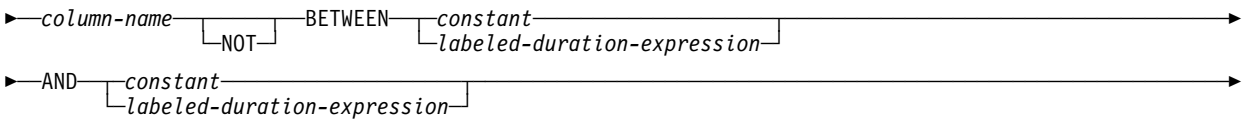
basic predicate:

Note: An exclamation point (!) is supported in place of the not symbol (¬). Therefore:

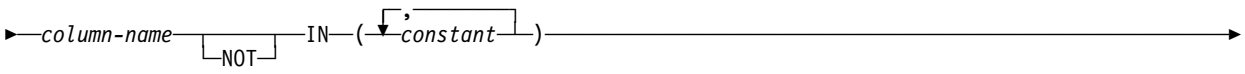
- != is equivalent to ¬=
- !> is equivalent to ¬>
- !< is equivalent to ¬<



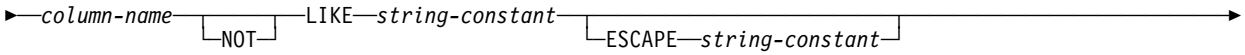
BETWEEN predicate:



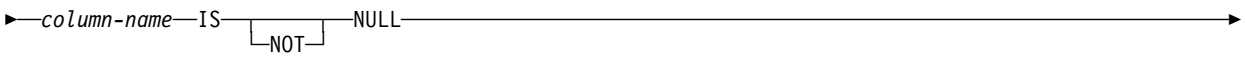
IN predicate:



LIKE predicate:



NULL predicate:



Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see “Control statement coding rules” on page 27.

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 to be reorganized. The name cannot be SYSUTILX if the database name specified is DSNDB01.

REUSE When used with SHRLEVEL NONE, specifies that REORG logically resets and reuses 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 will not be released if you use the REUSE parameter.

REUSE does not apply if you also specified SHRLEVEL REFERENCE or CHANGE.

PART *integer*
PART *integer1*:*integer2*

Identifies a partition 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 254.

integer Designates a single partition.

integer1:integer2 Designates a range of existing table space partitions from *integer1* through *integer2*.

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.

LOG Specifies whether records are logged during the reload phase of REORG. If the records are not logged, the table space is recoverable only after an image copy has been taken. If you specify COPYDDN, RECOVERYDDN, SHRLEVEL REFERENCE, or SHRLEVEL CHANGE, an image copy is taken during REORG execution.

YES Logs records 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 **LOG YES**.

You must specify LOG YES (explicitly or by default) for a LOB table space. Logging will occur only if the LOB table space was defined with the LOG YES attribute. If the LOB table space was defined with the LOG NO attribute, the LOB table space will be left in COPY pending status after the REORG.

NO Does not log records. Puts the table space in COPY pending status if either of these conditions is true:

- REORG is executed at the local site, and neither COPYDDN, SHRLEVEL REFERENCE, nor SHRLEVEL CHANGE are specified.

- REORG is executed at the remote site, and RECOVERYDDN is not specified.

SORTDATA Specifies that the data is to be unloaded by table space scan, then sorted in clustering order. Records always are sorted by the table in order to retain the clustering of records of the same table.

This option is recommended to improve performance unless one of the following is true:

- The data is in perfect clustering order and the REORG utility is used to reclaim space from dropped tables.
- The data set is very large and there is not enough DASD available for sorting.
- The longest possible composite record size is greater than 32760.

SORTDATA is ignored for some the catalog and directory table spaces; see “Reorganizing the catalog and directory” on page 315.

NOSYSREC Specifies that the output of sorting (if there is a clustering index and you specify SORTDATA) is the input to reloading, without using an unload data set. You can specify this option only if you specify REORG TABLESPACE, SORTDATA, 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 314 for additional information about using this option.

SORTKEYS Specifies that index keys will be sorted and indexes will be built in parallel during the SORTBLD phase to improve performance. This option is recommended if more than one index needs to be created.

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. A full image copy data set is created when REORG executes. In the row in the SYSIBM.SYSCOPY catalog table, the SHRLEVEL column is set to "R," as it would be for the COPY SHRLEVEL REFERENCE. The table space is not left 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, then no image copy is created at the local site.

COPYDDN(SYSCOPY) is assumed, and a DD statement for SYSCOPY is required if:

- You specify REORG SHRLEVEL REFERENCE or CHANGE, and do not specify COPYDDN
- A partition is in REORG pending (REORP) status

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 as for COPYDDN.

SHRLEVEL Specifies the method for performing the reorganization. The parameter following SHRLEVEL indicates the type of access 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 operates by unloading from the area 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), the following parameters cannot be specified: MAPPINGTABLE, MAXRO, LONGLOG, DELAY, and DEADLINE. If you omit SORTDATA, or specify UNLOAD PAUSE or UNLOAD ONLY, then you cannot specify NOSYSREC.

REFERENCE Specifies that reorganization operates by unloading from the area 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 applications' future access 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 used by REORG TABLESPACE" on page 307.

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, and DELAY.

SHRLEVEL REFERENCE is not allowed for a LOB table space.

CHANGE Specifies that reorganization operates by unloading from the area 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 applications' future access 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 used by REORG TABLESPACE" on page 307.

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.
- SORTDATA, NOSYSREC, SORTKEYS. Reorganization with CHANGE always operates as if these parameters were specified.
- UNLOAD. Reorganization with CHANGE always performs UNLOAD CONTINUE.

SHRLEVEL CHANGE is not allowed for a LOB table space.

DEADLINE Specifies the deadline for the switch phase to finish. If DB2 estimates that the SWITCH phase will not complete by the deadline, DB2 issues the messages that the -DISPLAY UTILITY command would issue and then terminates reorganization.

NONE Specifies that there is no deadline by which the SWITCH phase of log processing must finish.

The **default** is **NONE**.

timestamp *timestamp* specifies the deadline for the SWITCH phase of log processing to finish. 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 finish. The calculation is either based on CURRENT TIMESTAMP or CURRENT DATE. This deadline must not have already occurred when REORG is executed.

For example, to ensure that the SWITCH phase is complete by 6:30 AM two days from now, use the following expression:

```
DEADLINE CURRENT DATE + 2 DAYS + 6 HOURS + 30 MINUTES
```

If REORG SHRLEVEL REFERENCE or SHRLEVEL CHANGE terminates due to a DEADLINE specification, DB2 issues message DSNU374I with reason code 2, but does not set a restrictive status.

MAPPINGTABLE *table-name*

Specifies the name of the mapping table that REORG TABLESPACE uses 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 REORG TABLESPACE SHRLEVEL CHANGE, and you must create a mapping table and an index for it before running REORG TABLESPACE. The table must have the columns and the index that appear in the SQL statements described on page 304.

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 value specified with MAXRO.

The ALTER UTILITY command can change the value of MAXRO.

The **default** is **300** seconds. The value must be an integer.

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 will occur. The message also suggests that this would be an appropriate time to execute ALTER UTILITY to change the MAXRO value and thus cause REORG to finish. 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 just 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 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:

- There is a lot of SQL update activity during the log phase.
- The default behavior results in a large number of -911 SQL error messages.

LONGLOG Specifies the action that DB2 performs, after sending a message to the console, if the number of records that the next iteration of log process will process is not sufficiently lower than the number that the previous iterations processed. This situation means that reorganization's reading of the log is not catching up to applications' writing of the log quickly enough.

CONTINUE Specifies that until the time on the JOB statement expires, DB2 continues performing reorganization, including iterations of log processing, if the estimated time to perform an iteration exceeds the time specified with MAXRO.

A value of DEFER for MAXRO and a value of CONTINUE for LONGLOG together mean that REORG continues allowing access to the original copy of the area 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 will terminate reorganization after the delay specified by the DELAY parameter.

DRAIN Specifies that DB2 drains the write claim class after the delay 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 specified by the LONGLOG parameter.

integer is the number of seconds. The value must be an integer. The **default** is **1200**.

TIMEOUT Specifies the action 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 If a time out condition occurs, DB2 leaves the objects in a UTRO or UTUT state.

TERM If you specify the TERM option and a time out condition occurs, then DB2:

1. Issues an implicit TERM UTILITY command, causing the utility to end with a return code 8
2. Issues the DSNU590I and DSNU170I messages
3. Leaves the objects in a RW state.

OFFPOSLIMIT

The specified value is compared to the result of the calculation (NEAROFFPOSF + FAROFFPOSF) × 100 / CARDF for the specified partitions in SYSIBM.SYSINDEXPART for the explicit clustering indexes for every table in the specified table space. Alternatively, the values in SYSINDEXPART are checked for a single non-partitioned 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.

The **default** value is **10**.

INDREFLIMIT The specified value is compared to the result of the calculation (NEARINDREF + FARINDREF) × 100 / CARDF for the specified partitions in SYSIBM.SYSTABLEPART for the specified table space. Alternatively, the values in SYSTABLEPART are checked for a single non-partitioned 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 INDREFLIMIT value, REORG is performed or recommended. This option is valid for non-LOB table spaces only.

The **default** value is **10**.

REPORTONLY

If you specify this option, the REORG is only 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 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 subsystem.

CONTINUE Specifies that, after the data has been unloaded, the utility continues processing. An edit routine may 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 routine decoded. If you also specify DISCARD to a file, rows will be field procedure decoded, and SMALLINT, INTEGER, FLOAT, DECIMAL, DATE, TIME, and TIMESTAMP columns will be converted to external format. 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 ends. 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 you specify DISCARD, rows are decompressed and edit routine decoded. If you also specify DISCARD to a file, rows will be field procedure decoded, and SMALLINT, INTEGER, FLOAT, DECIMAL, DATE, TIME, and TIMESTAMP columns will be converted to external format. 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.

ONLY Specifies that after the data has been unloaded, the utility job ends and the status in SYSIBM.SYSUTIL corresponding to this utility ID is removed.

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 in SYSIBM.SYSUTIL corresponding to this utility ID is removed.

If you specify UNLOAD EXTERNAL with REORG TABLESPACE, rows are decompressed, edit routines decoded, field procedures are decoded, and SMALLINT, INTEGER, FLOAT, DECIMAL, DATE, TIME, and TIMESTAMP columns are converted to 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.

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 have to be compressed again.
- Existing compressed rows do not have to be expanded, unless indexes require it or SORTDATA is used.

Possible reasons for omitting 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.

For more information about specifying or omitting the KEEPDICTIONARY option, see “Compressing data” on page 158.

KEEPDICTIONARY is valid only if a compression dictionary exists and the table space or partition 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 regarding ESA data compression, see Section 2 (Volume 1) of *DB2 Administration Guide*.

STATISTICS Specifies the gathering of statistics for the table space or associated index, or both; the statistics are reported or stored in the DB2 catalog.

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.

TABLE Specifies the table for which column information is to be gathered. All tables must belong to the table space specified in the TABLESPACE option.

(ALL) Specifies that information is to be gathered for all columns of all tables in the table space.

SAMPLE *integer*
Indicates the percentage of rows to sample when collecting non-indexed column statistics. Any value from 1 through 100 can be specified. The default is *25*. The SAMPLE option is not allowed for LOB table spaces.

(*table-name***)**
Specifies the tables for which column information is to be gathered. The parentheses are required. If you omit the qualifier, the user identifier for the utility job is used.

If you specify more than one table, you must repeat the TABLE option.

COLUMN
Specifies columns for which column information is to be gathered.

You can only specify this option if you specify a 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.

(*column-name***, ...)**
Specifies the columns for which statistics are to be gathered. The parentheses are required.

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 specified in the TABLESPACE option.

(ALL) Specifies that the column information is to be gathered for all indexes defined on tables contained in the table space. The parentheses are required.

(*index-name***)** Specifies the indexes for which information is to be gathered. The parentheses are required.

REPORT Determines if a set of messages is generated to report the collected statistics.

NO Indicates that the set of messages is not output to SYSPRINT.

The **default** is **REPORT NO**.

YES Indicates that the set of messages is output to SYSPRINT. The messages generated are dependent on the combination of keywords (such as TABLESPACE, INDEX, TABLE, and COLUMN) 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 Tells whether the collected statistics are inserted into the catalog tables. UPDATE also allows you to select statistics used for access path selection or statistics used by database administrators.

ALL Indicates that all collected statistics will be updated in the catalog.

The **default** is **UPDATE ALL**.

ACCESSPATH Indicates that only the catalog table columns that provide statistics used for access path selection are updated.

SPACE Indicates that only the catalog table columns that provide statistics to help the database administrator assess the status of a particular table space or index are updated.

NONE Indicates that no catalog tables are updated with the collected statistics. This option is only valid when REPORT YES is specified.

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.

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 to concatenate together when collecting frequent values from the specified index. Specifying '3' means to collect frequent values on the concatenation of the first three key columns. The default is 1, which means collect frequent values on the first key column of the index.

COUNT Indicates the number of frequent values to be collected. Specifying '15' means collect 15 frequent values from the specified key columns. The default is *10*.

NOPAD Specifies that the variable-length columns in the unloaded or discarded records occupy the actual data length without additional padding. The unloaded records may 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.

The NOPAD option can only be specified with UNLOAD EXTERNAL or with DISCARD.

While the LOAD utility processes records with variable-length columns that were unloaded or discarded using the NOPAD option, these records can not be processed by applications that only process fields in fixed positions.

In order for the generated LOAD statement to provide a NULLIF condition for fields that are not in a fixed position, an input field definition is generated 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 generated for the EMP sample table would look similar to the LOAD statement shown in Figure 17 on page 295.

```
LOAD DATA INDDN SYSREC   LOG NO  RESUME YES
 EBCDIC CCSID(00500,00000,00000)
 INTO TABLE "DSN8610 "."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)
```

Figure 17 (Part 1 of 2). Sample LOAD statement generated by REORG TABLESPACE with the NOPAD keyword

```
 , "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 17 (Part 2 of 2). Sample LOAD statement generated by REORG TABLESPACE with the NOPAD keyword

PUNCHDDN *ddname*

Specifies the DD statement for a data set 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 last partition of a partitioned table space has had its limit key reduced.

DISCARDDN *ddname*

Specifies the DD statement for a discard data set, which is to hold copies of records that meet the DISCARD FROM TABLE ... WHEN specification.

ddname is the DD name.

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 **default** is **SYSDISC**.

DISCARD Specifies that records meeting 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 input, discarded records are saved in the associated data set.

DISCARD is valid only for SHRLEVEL NONE or SHRLEVEL REFERENCE. Discarding rows from a table that is part of a referential integrity set sets the CHECK pending status.

Do not specify DISCARD with the UNLOAD EXTERNAL or UNLOAD ONLY option.

FROM TABLE

The table space that is specified in REORG TABLESPACE can store more than one table. All tables are unloaded for UNLOAD EXTERNAL, and all tables might be subject to DISCARD. If you want to qualify the rows to be unloaded or discarded, you must use the FROM TABLE statement.

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.

WHEN

The WHEN clause tells 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.

selection condition

A *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 or DISCARD. 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, OR, NOT) 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.

predicate
A *predicate* specifies a condition that is true, false, or unknown about a row.

labeled-duration-expression
A *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) and optionally contains arithmetic operations of addition or subtraction expressed by a number 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 performs 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. 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. Here 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 will affect 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:

```
CURRENT DATE + 1 YEAR + 1 DAY
```

To subtract one year, one month, and one day from a date, specify:

```
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.

basic predicate
A *basic predicate* compares 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 = constant	column is equal to constant or labeled duration expression.
column < > constant	column is not equal to constant or labeled duration expression.
column > constant	column is greater than constant or labeled duration expression.
column < constant	column is less than constant or labeled duration expression.
column > = constant	column is greater than or equal to constant or labeled duration expression.
column < = constant	column is less than or equal to constant or labeled duration expression.
column ¬ = constant	column is not equal to constant or labeled duration expression.
column ¬ > constant	column is not greater than constant or labeled duration expression.
column ¬ < constant	column is not less than constant or labeled duration expression.

For ASCII table spaces, the constant must be specified in hexadecimal.

BETWEEN predicate

The BETWEEN predicate determines whether a given value lies between two other given values specified in ascending order. Each of the predicate's two other forms has an equivalent search condition, as shown below:

The predicate: *column* BETWEEN *value1* AND *value2*

is equivalent to: (*column* >= *value1* AND *column* <= *value2*)

The predicate: *column* NOT BETWEEN *value1* AND *value2*

is equivalent to: NOT(*column* BETWEEN *value1* AND *value2*)

and therefore also to: (*column* < *value1* OR *column* > *value2*)

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

The IN predicate compares a value with a set of values. In the IN predicate, the second operand is a set of one or more values specified by constants.

The predicate: *value1* IN (*value1*, *value2*,..., *valuen*)

is equivalent to: (*value1* = *value2* OR ... OR *value1* = *valuen*)

The predicate: *value1* NOT IN (*value1*, *value2*,..., *valuen*)

is equivalent to: (*value1* ¬= *value2* AND ... AND *value1* ¬= *valuen*)

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

The *LIKE predicate* qualifies strings that have a certain pattern. Specify the pattern with a string in which the underscore and percent sign characters have special meanings.

Let *x* denote the column 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* 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*. Matching the pattern is described below.
- When *x* or *y* is null, the result of the predicate is unknown.
- When *y* is empty and *x* is not, the result of the predicate is false.
- When *x* is empty and *y* is not, 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 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 a SBCS character and *y* is interpreted as SBCS data regardless of is subtype. The rules for **mixed data patterns** are described on page 301.

Within the pattern, a percent sign or underscore character can have a special meaning, or it can represent the literal occurrence of a percent sign or underscore character. 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 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 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.

A more rigorous description of strings and patterns follows

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

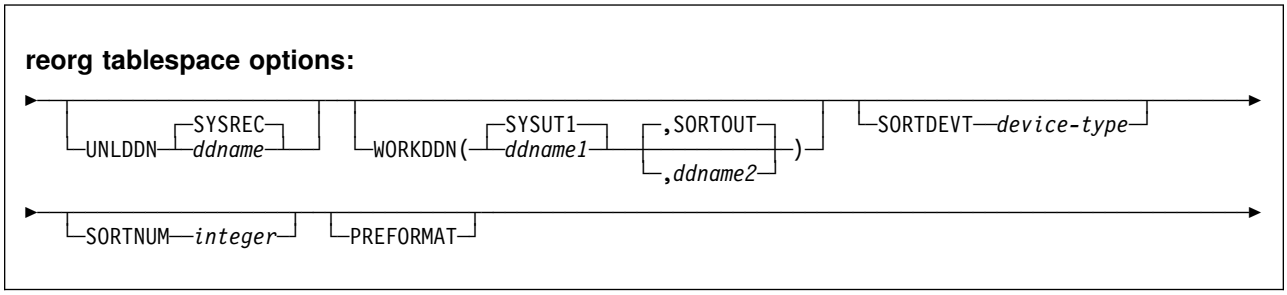
The *NULL predicate* tests for null values.

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.

reorg tablespace options

For the descriptions of keywords and parameters included within *reorg tablespace options*, see page 302.

REORG TABLESPACE options syntax



Option descriptions for REORG TABLESPACE options

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see "Control statement coding rules" on page 27.

UNLDDN *ddname* Specifies the DD name of the unload data set.

ddname is the DD name of the unload data set.

The **default** is **SYSREC**.

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

ddname specifies the DD statement for a temporary data set used for intermediate output.

ddname1 Is the DD name of the temporary work file for sort input. DB2 requires a work data set for sort input for tables with indexes, unless you specify SORTKEYS.

The **default** is **SYSUT1**.

ddname2 Is the DD name of the temporary work file for sort output. DB2 requires a work data set for sort output, unless you specify SORTKEYS.

The **default** is **SORTOUT**.

Even though WORKDDN is an optional keyword, a DD card for the sort output data set is required in the JCL unless you specify SORTKEYS. If you do not specify WORKDDN, or if you specify it without a *ddname2*, the JCL must have a DD card with the name SORTOUT. If *ddname2* is given, then a DD card must be supplied that matches it.

WORKDDN is ignored for the catalog and directory table spaces listed in "Reorganizing the catalog and directory" on page 315.

SORTDEVT *device-type*

Specifies the device type for temporary data sets to be dynamically allocated by DFSORT.

device-type is the device type, and can be any acceptable to the DYNALLOC parameter of the SORT or OPTION control statement for DFSORT.

If you omit SORTDEVT and require a sort of the index keys, you must provide the DD statements that the sort program needs for the temporary data sets.

SORTDEVT is ignored for the catalog and directory table spaces listed in “Reorganizing the catalog and directory” on page 315.

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

integer is the number of temporary data sets.

If you omit SORTDEVT, SORTNUM is ignored. If you use SORTDEVT and omit SORTNUM, no value is passed to DFSORT. It is allowed to take its own default.

SORTNUM is ignored for the catalog and directory table spaces listed in “Reorganizing the catalog and directory” on page 315.

PREFORMAT Specifies that the remaining pages are preformatted up to the high allocated RBA in the table space and index spaces associated with the table 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 its corresponding partitioning index space.

PREFORMAT is ignored if you specify UNLOAD ONLY or UNLOAD EXTERNAL.

For more information on PREFORMAT, see “Improving performance with LOAD or REORG PREFORMAT” on page 162.

Instructions for running REORG TABLESPACE

To run REORG TABLESPACE, you must:

1. Read “Before running REORG TABLESPACE” on page 304 in this chapter.
2. Prepare the necessary data sets, as described in “Data sets used by REORG TABLESPACE” on page 307.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27. (For examples of JCL for REORG TABLESPACE, see “Sample control statements” on page 335.)
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 310. (For a complete description of the syntax and options for REORG TABLESPACE, see “Syntax and options of the control statement” on page 279.)
5. Check the compatibility table in “Concurrency and compatibility” on page 329 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 325.

7. Run REORG TABLESPACE.

See “Chapter 2-1. Invoking DB2 online utilities” on page 27 for an explanation of ways to execute DB2 utilities.

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, DSNDB01.DBD01, and DSNDB01. SYSUTILX catalog table spaces, REORG scans logs to verify that an image copy is available. If the scan of logs does not find an image copy, DB2 will request archive logs.

Region size: The recommended minimum region size is 4096K.

Mapping table and SHRLEVEL CHANGE: Before running REORG TABLESPACE with SHRLEVEL CHANGE, you must create a **mapping table** and an 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:

```
CREATE TABLESPACE table-space-name SEGSIZE integer
```

The number of rows in the table should not exceed 110% of the number of rows in the table space or partition to be reorganized. The mapping table must have only the columns and the index 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 TARGET_XRID column must be specified as CHAR(9) even though the RIDs are still 5 bytes long.

You must have DELETE, INSERT, SELECT, and UPDATE authorization on the mapping table.

You can run more than one REORG SHRLEVEL CHANGE job concurrently, either on separate table spaces or different partitions of the same table space. When running concurrently, each REORG must have a separate mapping table. The mapping tables need not 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 tries to access the same mapping table at the same time, one of the REORGs will fail.

For a sample of REORG with SHRLEVEL CHANGE and a sample mapping table and index, see job sample DSNTEJ1 as described in *DB2 Installation Guide*.

User-managed data sets and SHRLEVEL REFERENCE and CHANGE: If a table space, partition, or index to be reorganized resides in user-managed data sets, then before executing the REORG utility with SHRLEVEL REFERENCE or SHRLEVEL CHANGE, you must create shadow data sets with the names and

attributes described in Section 2 (Volume 1) of *DB2 Administration Guide*. The data sets must already exist when you execute REORG.

The names have the form `catname.DSNDBx.dbname.psname.S0001.Annn`. Define the data sets as LINEAR and use SHAREOPTIONS(3,3). An efficient method for defining shadow data sets specifies MODEL, so the shadow is created like the original. For example:

```
DEFINE CLUSTER +
       (NAME('catname.DSNDBC.dbname.psname.S0001.A001') +
       MODEL('catname.DSNDBC.dbname.psname.I0001.A001')) +
      DATA                                               +
       (NAME('catname.DSNDBD.dbname.psname.S0001.A001') +
       MODEL('catname.DSNDBD.dbname.psname.I0001.A001') )
```

When reorganizing an entire table space, you must create the shadow data sets for the table space and all indexes.

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. You can create the shadows ahead of time for DB2-managed data sets, and it is strongly recommended that you do so for the shadow data set of the logical partition of nonpartitioning indexes.

Regardless of whether the area being reorganized resides in user-managed or DB2-managed data sets, data sets with names that have the form `catname.DSNDBx.dbname.psname.T0001.Annn` must not already exist when you execute REORG with SHRLEVEL REFERENCE or SHRLEVEL CHANGE.

Estimating the size of pre-allocated data sets: If you have not changed the value of FREEPAGE or PCTFREE, the amount of space required for a shadow data set should be comparable to the amount of space required for the original data set. However, for REORG PART, the space required for the shadow of the logical partition of a nonpartitioning index is approximately equal to the percentage of space the partition occupies in the entire table space. For example, a partitioned table space with 100 partitions and data relatively evenly balanced across the partitions needs a shadow for the logical partition roughly 1 percent the size of the original nonpartitioning index.

Pre-allocating shadow data sets for REORG PART: By creating the shadow data sets for REORG PART ahead of time, even for DB2-managed data sets, you prevent possible over-allocation of DASD during REORG processing. When reorganizing a partition, you must create the shadow data sets for the partition of the table space and 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, for each nonpartitioning index that resides in user-defined data sets, a shadow data set for a copy of the logical partition of the index. The name for this shadow data set has the form `catname.DSNDBx.dbname.psname.S0mmm.Annn`.

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 will be contained in the single shadow data set. The name for this shadow data set must have the form `catname.DSNDBx.dbname.psname.S0mmm.Annn`, where *mmm* is the first partition in the range specification.

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 held.

Data sharing considerations for REORG: You must 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.

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
- The clustered index is in the REBUILD pending status, and the data is unloaded by the cluster index method.

Similarly, you cannot reorganize a single table space partition if:

- 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 table space is a subset of a range of partitions that are in REORG pending status; you must reorganize the entire range to reset the restrictive status.

There is one RECOVER pending restrictive state:

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 not in RECP. RECP can be reset by recovering only the single logical partition.

There are three REBUILD pending restrictive states:

RBDP REBUILD pending status (RBDP) is set on a physical or logical index partition. The individual physical or logical partition is inaccessible and must be rebuilt using the REBUILD INDEX utility.

PSRBD Page set REBUILD pending (PSRBD) is set for nonpartitioning indexes. The entire index space is inaccessible and must be rebuilt 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 is made available again when the affected partitions are rebuilt using the REBUILD INDEX utility.

For information about resetting the REBUILD pending and RECOVER pending states, see Table 93 on page 531 and Table 92 on page 530.

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 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 until the CHECK pending status is cleared. See "CHECK pending status" on page 528 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 used by REORG TABLESPACE

Table 53 describes the data sets used by REORG TABLESPACE. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 53 (Page 1 of 2). Data sets used by REORG TABLESPACE

Data Set	Description	Required?
SYSDISC	Contains discarded records from REORG DISCARD; optional for REORG The default DD name is SYSDISC.	DISCARDDN
SYSIN	Input data set containing the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
SYSPUNCH	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.	PUNCHDDN
Unload data set	Data set for the unloaded data and the data set to be loaded by the RELOAD phase. The data set is identified by the DD statement named in the UNLDDN keyword or by 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.	Yes[2]
Copies	From 1 to 4 output data sets to contain the image copies. Their DD names are specified with the COPYDDN and RECOVERYDDN options of the utility control statement.	Yes[3]
Work data sets	Temporary data sets for sort input and output. The DD names have the form DATAWK*nn*.	Yes[4]

Table 53 (Page 2 of 2). Data sets used by REORG TABLESPACE

Data Set	Description	Required?
Work data sets	Temporary data sets for sort input and output. The DD names have the form SORTWK*nn*.	Yes[5]
Work data sets	Two temporary data sets for sort input and sort output. Their DD names are specified 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.	No[1]

Note:

[1] Not required if SORTKEYS is used; otherwise, required for tables with indexes.

[2] Required unless NOSYSREC or SHRLEVEL CHANGE is specified.

[3] Required if COPYDDN, RECOVERYDDN, SHRLEVEL REFERENCE, or SHRLEVEL CHANGE is specified.

[4] Required if NOSYSREC or SHRLEVEL CHANGE is specified but SORTDEVT is not specified.

[5] Required if any indexes exist and SORTDEVT is not specified.

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

Table space The name of the table space to be reorganized. It is named in the control statement and is accessed through the DB2 catalog.

Calculating the size of the unload data set: The size required for the unload data set varies depending on the options used for REORG.

1. If you use REORG with UNLOAD PAUSE or CONTINUE and with KEEPDICTIONARY (assuming a compression dictionary already exists), the size of the unload data set, in bytes, can be roughly calculated as the VSAM hi-used RBA for the table space. The hi-used RBA can be obtained from the associated VSAM catalog. For SHRLEVEL CHANGE, add (number of records * 11) bytes to the VSAM hi-used RBA.
2. If you use REORG UNLOAD ONLY, or UNLOAD PAUSE or CONTINUE without KEEPDICTIONARY, the size of the unload data set, in bytes, can be calculated as the maximum row length multiplied by the 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 there are multiple tables in the table space, the formula is:

   ```
   Sum over all tables (row length × number of rows)
   ```

   For SHRLEVEL CHANGE, also add:

   ```
   (21 × ((NEARINDREF + FARINDREF) × 1.1))
   ```

   where:

   NEARINDREF Value obtained from the NEARINDREF column of the

SYSIBM.SYSTABLEPART catalog table[4].

FARINDREF Value obtained from the FARINDREF column of the SYSIBM.SYSTABLEPART catalog table[4].

3. If you have variable length fields, the calculation in 2 on page 308 might give you excessive space. Use the average uncompressed row length multiplied by the number of rows.

For certain table spaces in the catalog and directory, the unload data set for the table spaces will have a different format. The calculation for the size of this data set is as follows:

```
data set size in bytes = (28 + longrow) × numrows
```

where:

longrow Length of the longest row in the table space

numrows The number of rows in the data set

The length of the row is calculated:

```
Sum of column lengths + 4 bytes for each link
```

The length of the column is calculated:

```
Maximum length of the column + 1 (if nullable) + 2 (if varying
  length)
```

See “Reorganizing the catalog and directory” on page 315 for more information about reorganizing catalog and directory table spaces.

Calculating the size of the work data sets: When reorganizing an index space or a table space with indexes, you need a non-DB2 sequential work data set unless you specify the SORTKEYS keyword. That data set is identified by the DD statement named in the WORKDDN option. During the RELOAD phase, the index keys and the data pointers are unloaded to the work data set. This data set is used to update the index data pointers after the data has been moved. It is required only during the execution of REORG.

To calculate the approximate size (in bytes) of both WORKDDN data sets SORTOUT and SYSUT1, follow these steps:

1. For each table, calculate the number of keys:

   ```
   number of keys = (#tablerows × #indexes)
   ```

   where:

   #tablerows Number of records in the table.

   #indexes Number of indexes defined on the table.

   For SHRLEVEL CHANGE, *#indexes* should count the number of indexes on the table, plus 1 for the mapping index.

2. Add the sums obtained in step 1.

   For SHRLEVEL CHANGE, also add:

   ```
   ((NEARINDREF + FARINDREF) × 1.1)
   ```

[4] The accuracy of the data set size calculation depends on recent information in the SYSTABLEPART catalog table.

where:

NEARINDREF Value obtained from the NEARINDREF column of the SYSIBM.SYSTABLEPART catalog table[4].

FARINDREF Value obtained from the FARINDREF column of the SYSIBM.SYSTABLEPART catalog table[4].

3. Multiply the sum in step 2 by the longest key length plus 9. When determining the longest key length, remember that the length of the mapping index is 21 bytes.

Allocating twice the space used by the input data sets is usually adequate for the sort work data sets. For compressed data, double again the space allocated for the sort work data sets if you use the following REORG options:

- UNLOAD PAUSE without KEEPDICTIONARY
- UNLOAD CONTINUE without KEEPDICTIONARY

Two or three large SORTWK*nn* data sets are preferable to 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 to define a destination for messages issued by DFSORT during the SORT phase of REORG. The default DD statement used by DB2I and the %DSNU CLIST command and by the DSNUPROC procedure is:

```
//UTPRINT DD  SYSOUT=A
```

Creating the control statement

See "Syntax and options of the control statement" on page 279 for REORG TABLESPACE syntax and option descriptions. See "Sample control statements" on page 335 for examples of REORG TABLESPACE usage.

Instructions for specific tasks

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

Product-sensitive Programming Interface

Determining when an object should be reorganized

You can determine when to run REORG for non-LOB table spaces and indexes by using the OFFPOSLIMIT, INDREFLIMIT catalog query options. If you specify the REPORTONLY option, REORG will produce a report detailing if 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 performed or recommended.
2 REORG performed or recommended.

Alternatively, information from the SYSTABLEPART and SYSINDEXPART catalog tables can tell you which table spaces and indexes qualify for reorganization. This information can also be used to determine when the DB2 catalog table spaces require reorganization. For table spaces SYSDBASE, SYSVIEWS, and SYSPLAN of the catalog, the value for columns FAROFFPOS and NEAROFFPOS of SYSINDEXPART should not be used when determining 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, “Resetting an advisory or restrictive status” on page 527 for more information.

Information from the SYSTABLEPART catalog table can also tell you how well DASD space is being used. If you want to find the number of varying-length rows relocated to other pages because of an update, run RUNSTATS and issue this statement:

```
SELECT CARD, NEARINDREF, FARINDREF
  FROM SYSIBM.SYSTABLEPART
  WHERE DBNAME = 'XXX'
  AND TSNAME = 'YYY';
```

A large number (relative to previous values 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 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 (XXX.YYY):

```
SELECT NEAROFFPOSF, FAROFFPOSF
  FROM SYSIBM.SYSINDEXPART
  WHERE IXCREATOR = 'index_creator_name'
  AND IXNAME = 'index_name';
```

There are several indicators available to signal a time for reorganizing the table spaces. A large value for FAROFFPOSF might indicate that clustering is degenerating. Reorganizing the table space would improve performance.

A large value for NEAROFFPOSF might indicate also that reorganization might improve performance. However, in general it is not as critical a factor as FAROFFPOSF.

FAROFFPOSF and NEAROFFPOSF do not have performance considerations for certain DB2 catalog tables.

DSNDB06.SYSDBASE
DSNDB06.SYSDBAUT
DSNDB06.SYSGROUP
DSNDB06.SYSPLAN
DSNDB06.SYSVIEWS
DSNDB01.DBD01

For any table, the REORG utility repositions rows into the sequence of the key of the clustering index defined on that table. If you specify the SORTDATA option of the REORG utility, the data is unloaded using a sequential scan. If you do not specify the SORTDATA option, REORG uses the clustering index to unload the data.

For nonclustering indexes, the statistical information 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 CLUSTERING and CLUSTERED in SYSINDEXES and to NEAROFFPOS and FAROFFPOS in SYSINDEXPART.

In general, it is a good practice to 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 you have to your data during reorganization:

- REORG with SHRLEVEL NONE, the default, reloads the reorganized data into the original area 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 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 being reorganized. For REORG TABLESPACE SHRLEVEL CHANGE, a mapping table maps between RIDs in the original copy of the table space or partition and RIDs in the shadow copy; see page 304 for instructions on creating the mapping table. Applications can read and write 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 specified by MAXRO. If this condition is met, the next iteration will be 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 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 specified by DELAY and then performs the action specified by LONGLOG.

Operator actions: LONGLOG specifies the action that DB2 performs if log processing is not catching up. See “Option descriptions” on page 282 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 you may take:

- 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 increases the likelihood that log processing will catch up. 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 allocated to reorganization and to applications. This can increase the likelihood that log processing will catch up. 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 can increase the likelihood that log processing will catch up. 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 will be less than or equal to the time specified in the MAXRO phrase.
- 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 specify SORTDATA, SHRLEVEL REFERENCE, or SHRLEVEL NONE, and 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:

- For REORG TABLESPACE SORTDATA NOSYSREC, DB2 assumes there is a clustering index present.
- For REORG TABLESPACE SHRLEVEL CHANGE, REORG omits the unload data set, even if you omit NOSYSREC, unless there is no explicit clustering index.
- 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 using the contents of the unload data set. However, if you specify 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 in addition to any image copies you create during or after REORG.

Unloading without reloading

REORG can unload data without continuing and without leaving 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 may add a validation routine to a table. On reloading, all the rows will be checked by the validation procedure.

REORG UNLOAD ONLY should **not** be used for data propagation. When you specify the UNLOAD ONLY option, REORG only unloads what physically resides in the base table space; LOBs are not unloaded. For purposes of data propagation, you should use REORG UNLOAD EXTERNAL instead.

Reclaiming space from dropped tables

Reorganization omits tables that have been previously dropped, reclaiming the space they acquired. See “Reclaiming space in the DBD” on page 199 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, it restores from the log.

However, if you use REORG SHRLEVEL NONE LOG NO, RECOVER cannot restore 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 fall back recovery.

To create a new recovery point, it is strongly recommended that immediately following an ALTER INDEX operation that modifies key values, you either:

- Run REORG with COPYDDN and SHRLEVEL NONE specified, or
- Take a full image copy immediately after REORG completes.

If you performed a REORG to turn off REORG pending status (REORP), you should also take an inline image copy, or run the COPY utility. Image copies taken prior to resetting the REORG pending status cannot be used for recovery to current RBA or LRSN.

Successful REORG LOG NO processing inserts an SYSIBM.SYSCOPY row with ICTYPE='W' for each 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.

Reorganizing the catalog and directory

You can run REORG TABLESPACE on the table spaces in the catalog database (DSNDB06) and the SCT02, SPT01, and DBD01 table spaces in the directory database (DSNDB01).

Attention: You must take a full image copy before and after reorganizing any catalog or directory object.

When you REORG 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.

When to run REORG on the catalog and directory: You should not need to run REORG TABLESPACE on the catalog and directory table spaces as often as you do on user table spaces. The statistics collected by RUNSTATS that you use to determine if a REORG is required for a user table space can also be used for the catalog table spaces. The only difference is the information in the columns NEAROFFPOS and FAROFFPOS in table SYSINDEXPART. These columns can tolerate a higher value before a reorganization is needed if the table space is DSNDB06.SYSDBASE, DSNDB06.SYSVIEWS, or DSNDB06.SYSPLAN. When it is determined that any of the following catalog table spaces require reorganization, you should also reorganize the corresponding directory table space:

Catalog Table Space	Directory Table Space
DSNDB06.SYSDBASE	DSNDB01.DBD01
DSNDB06.SYSPLAN	DSNDB01.SCT02
DSNDB06.SYSPKAGE	DSNDB01.SPT01

Fragmentation and wasted space in the catalog table spaces affect the performance of user queries against the catalog and performance of DB2 functions.

Associated directory table spaces: When certain catalog table spaces are reorganized, you should reorganize the associated directory table space as well.

Limitations for reorganizing the catalog and directory:

- You cannot reorganize DSNDB01.SYSUTILX or DSNDB01.SYSLGRNGX.
- The UNLOAD ONLY and LOG YES options are not allowed for catalog and directory table spaces
- The WORKDDN, SORTDATA, SORTDEVT, SORTNUM, SORTKEYS, COPYDDN, and RECOVERYDDN options are ignored for the following catalog and directory table spaces:

  DSNDB06.SYSDBASE
  DSNDB06.SYSDBAUT
  DSNDB06.SYSGROUP
  DSNDB06.SYSPLAN
  DSNDB06.SYSVIEWS
  DSNDB01.DBD01

- REORG TABLESPACE with SHRLEVEL REFERENCE or CHANGE cannot operate on the following catalog and directory table spaces:

  DSNDB06.SYSDBASE
  DSNDB06.SYSDBAUT
  DSNDB06.SYSGROUP
  DSNDB06.SYSPLAN
  DSNDB06.SYSVIEWS
  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.

Changing data set definitions

If the table 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. The user effectively changes the characteristics of a user-managed data set by specifying the desired new characteristics when creating the shadow data set; see page 304 for more information about user-managed data sets. In particular, placing the original and shadow data sets on different DASD 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 using the phase restart or current restart. The REORG statement must not be altered.

The SYSIBM.SYSUTIL record for the REORG utility remains in "stopped" status until REORG is restarted or terminated.

While REORG is interrupted by PAUSE, you may 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 TABLESPACE.

Building a compression dictionary

The compression dictionary is built during the UNLOAD phase. 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

When you specify SORTDATA on your REORG statement, DB2 estimates how many rows are to be sorted and passes this information to DFSORT on the parameter FILSZ, letting DFSORT dynamically allocate the necessary sort workspace.

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, DFSORT might fail and cause an abend.

If compression is defined for the table space, REORG doubles the estimated FILSZ so that DFSORT allocates enough space to expand the compressed records during the UNLOAD phase.

You can override this dynamic allocation of sort workspace in two ways:

- Allocate the sort work data sets with SORTWK*nn* DD statements in your JCL
- Override DB2's row estimate in FILSZ using control statements passed to DFSORT. However, using control statements overrides size estimates passed to DFSORT in all invocations of DFSORT in the job step, including sorting keys to build indexes, and any sorts done in any other utility executed in the same step. The result might be reduced sort efficiency or an abend due to an out of space condition.

Rebalancing partitions using REORG

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 REORG:

```
REORG TABLESPACE DSN8S61E PART 2:3
 SORTDATA
 NOSYSREC
 SORTKEYS
 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 on 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 non-restricted 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 part, 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 subsequently had an ALTER INDEX statement reduce the limit key for the last partition of the table space. Otherwise, REORG will terminate issuing 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.

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 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 REORG processing.

The SYSCOPY record 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 an W if the image copy was produced by REORG TABLESPACE LOG(NO). The data set 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 will be out of sequence and might be repeated

The total number of duplicate pages will be small, with a negligible effect on the space required for the data set. One exception to this guideline is in the case of running REORG SHRLEVEL CHANGE, where the number of duplicate pages will vary with the number of records applied during the LOG phase.

Improving performance

To improve REORG performance:

- We strongly recommend that you specify SORTDATA on your REORG statement unless your data set is very large, and you do not want to allocate the extra DASD required by DFSORT. In this case, DB2 unloads the data by table space scan and executes DFSORT to sort the data into clustering order. SORTDATA is useful if either:
  - Your table's CLUSTERRATIOF is less than 95% or
  - FAROFFPOS/CARD for your table is greater than 5%

In general, the lower the CLUSTERRATIOF of your tables, the greater the performance improvement of REORG when specifying SORTDATA.

- 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 of the entire table space. However, the elapsed time of reorganizing the entire table in parallel may be significantly less than it would be for a single REORG job.
- Specify SORTKEYS on your REORG statement to sort index keys in parallel with the reload and build phases. This option passes index keys in sort to memory rather than writing them to sort input and output data files. In addition to improving performance, this option reduces work space requirements because the SYSUT1 and SORTOUT data sets are not required. This option is recommended if more than one index needs to be created. REORG will implicitly use SORTKEYS if you specified SHRLEVEL CHANGE. However, if a job using SORTKEYS abends in the RELOAD, SORT, BUILD, or SORTBLD phase, it can only be restarted at the beginning of the reload phase.

  Use parallel index build for table spaces or partitions with more than one index defined. For more information, see "Building indexes in parallel for REORG TABLESPACE" on page 321.
- Specify NOSYSREC on your REORG statement. See "Omitting the output data set" on page 314 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 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.

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.

Performance implications with SHRLEVEL CHANGE: Under certain circumstances, the log records used by REORG SHRLEVEL CHANGE 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 being reorganized uses data compression
  - The area is a partitioned table space, and at least one partition uses data compression

Building indexes in parallel for REORG TABLESPACE: Use parallel index build to reduce 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 process each index; one subtask sorts extracted keys while the other subtask builds the index. REORG TABLESPACE begins building each index as soon as the corresponding sort emits its first sorted record. For more information about improving index key sort performance, see “Improving performance” on page 319.

Figure 18 shows a REORG TABLESPACE flow with parallel index build, which requires SORTKEYS. DB2 starts multiple subtasks to sort index keys and build indexes in parallel. If you specified 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.

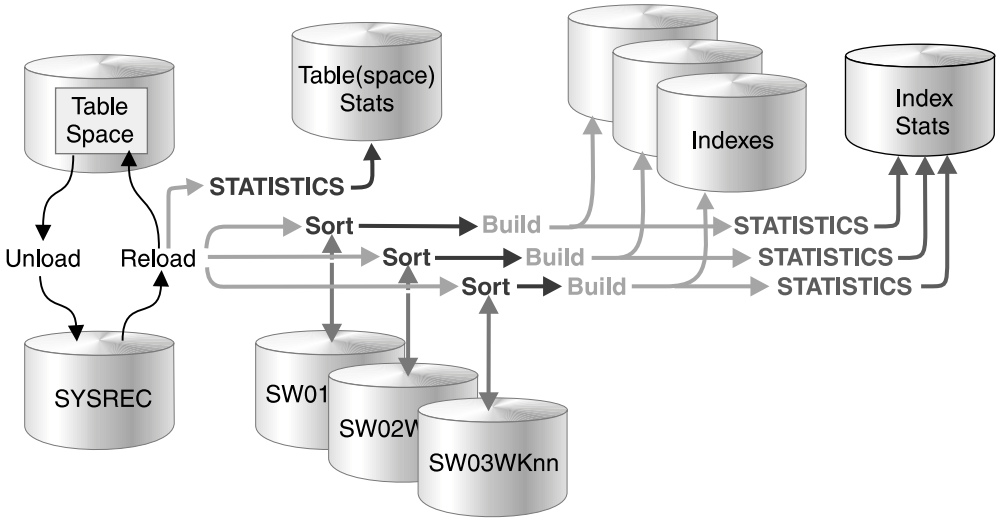


Figure 18. Building indexes using parallel index build

REORG TABLESPACE uses parallel index build if all of the following conditions are true:

- There is more than one index to be built (including the mapping index for SHRLEVEL CHANGE).
- You specify either SORTKEYS or SHRLEVEL CHANGE in the utility statement.
- You either allow the utility to dynamically allocate the data sets needed by SORT, 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 and message data sets.

1. Specify the SORTKEYS and SORTDEVT keywords 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: Allows you to control allocation of sort work data sets, while REORG TABLESPACE allocates message data sets.

1. Specify the SORTKEYS keyword in the utility statement.
2. Provide DD statements with DDNAMEs in the form SW*nn*WK*mm*.
3. Allocate UTPRINT to SYSOUT.

Method 3: Allows the most control over rebuild processing; you must specify both sort work and message data sets.

1. Specify the SORTKEYS keyword in the utility statement.
2. Provide DD statements with DDNAMEs in the form SW*nn*WK*mm*.
3. Provide DD statements with DDNAMEs in the form UTPRIN*nn*.

Data sets used: If you select Method 2 or 3 above, use the information provided here along with "Determining the number of sort subtasks," "Allocation of sort subtasks" on page 323 , and "Estimating the sort work file size" on page 323 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 DASD free space.
- To limit the number of utility subtasks used to build indexes.

The DDNAMEs SW*nn*WK*mm* define the sort work data sets used during utility processing. *nn* identifies the subtask pair, while *mm* identifies one or more data sets to be used by that subtask pair. For example:

SW01WK01 The first sort work data set used by the subtask building the first index.

SW01WK02 The second sort work data set used by the subtask building the first index.

SW02WK01 The first sort work data set used by the subtask building the second index.

SW02WK02 The second sort work data set used by the subtask building the second index.

The DDNAMEs UTPRIN*nn* define the sort work message data sets used by the utility subtask pairs. *nn* identifies the subtask pair.

Determining the number of sort subtasks: The maximum number of utility subtask pairs started for parallel index build is equal to the number of indexes 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 sort work data set groups allocated.
- The number of subtask pairs equals the number of message data sets allocated.

- If you allocate both sort work and message data set groups, the number of subtask pairs equals the smallest number of data sets allocated.

Allocation of sort subtasks: REORG TABLESPACE attempts to assign one sort subtask pair for each index 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, so 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 will need to know the size and number of keys present in all of the indexes being processed by the subtask in order to calculate each sort work file size. After you've determined which indexes are assigned to which subtask pairs, use the following formula to calculate the space required:

2 × (longest index key + 9) × (number of keys extracted)

longest key The length of the longest index key that will be processed by the subtask. For SHRLEVEL CHANGE, the mapping index key length is 21.

number of keys The number of keys from all indexes to be sorted that will be processed by the subtask.

Considerations for running REORG

This section discusses additional points to keep in mind when running REORG. See Appendix C, “Resetting an advisory or restrictive status” on page 527 for information on resetting REORG pending status.

Sorting data in clustering order

When you specify SORTDATA:

- If an explicit clustering index exists on any table in the table space being reorganized, rows of the table space are unloaded in physical sequence. (If the object being reorganized is a partition, rows of that partition are unloaded in physical sequence.) DFSORT then uses the clustering index key to sort the rows. If any other table in the table space has no explicit clustering index, the key of the implicit clustering index (if one exists) is used for the sort.
- If no explicit clustering index exists, SORTDATA is ignored.
- If the largest possible composite record to be sorted exceeds 32760 bytes in length, which is the maximum record size for a BSAM data set, SORTDATA is ignored if you specified SHRLEVEL NONE or REFERENCE (REORG cannot operate if a table has a clustering index and you specified SHRLEVEL CHANGE). This condition can occur only when the table space contains pages of 32 KB size. The largest possible composite record in the table space may be calculated by the following formula:

  ```
  max(K) + max(R + E) + 18 bytes for NONE or REFERENCE or 29 bytes for
  CHANGE
  ```

where:

max(K) = number of bytes in the longest possible clustering key (K)

max(R + E) = number of bytes in the longest possible record (R) plus edit procedure work area (E). E = 10 bytes if an edit procedure is used, otherwise E = 0 bytes.

When SORTDATA is not specified:

- If an explicit clustering index exists, segmented table spaces are unloaded using that index
- If an explicit clustering index does not exist, the table space is unloaded by table. Multi-table simple table spaces are unloaded by table space scan, in which case rows are reloaded in the same order that they were unloaded.

Methods of unloading data

Data is unloaded by one of three methods:

- *Table space scan with sort*: Chosen if you specified the SORTDATA option, and at least one table in the table space has an explicit clustering index.
- *Table space scan*: Chosen for simple table spaces that contain more than one table, or contain one table but do not have an explicit clustering index.
- *Clustering index*: Always chosen for partitioned table spaces (unless you specified the SORTDATA option); chosen for simple table spaces that contain one table and have an explicit clustering index; and chosen for tables in a segmented table space that have an explicit clustering index.

Encountering an error in the RELOAD phase

Failure during the RELOAD phase (after the data has been unloaded and data sets have been deleted, but before the data has been 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 do allocate new data sets, do alter the table space, or do 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 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 267.

Reorganizing segmented table spaces

If the target table space is segmented, REORG unloads and reloads by table.

If an explicit clustering index exists on a table in a segmented table space, that table is unloaded in clustering sequence. If NO explicit clustering index exists, the table is unloaded in physical row and segment order.

For segmented table spaces, REORG does NOT normally have to reclaim space from dropped tables. Space 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, then REORG reclaims the space for dropped tables.

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 checks the count of the records that were actually loaded against the count of the records that were unloaded. If the counts do not the match, the actions taken depend on the UNLOAD option you specified on the original job:

- If you specified UNLOAD PAUSE, REORG sets return code 4 and continues processing the job.
- If you specified UNLOAD CONTINUE, DB2 issues an error message and abends 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. A LOB table space that was defined with LOG YES or LOG NO will affect logging while reorganizing a LOB column. Table 25 on page 170 shows the logging output and LOB table space effect, if any.

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

If you terminate REORG TABLESPACE with the TERM UTILITY command during the UNLOAD phase, objects have not yet been changed and the job can be rerun.

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 are left 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 not yet built are left 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 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 there is a problem in renaming to the original data sets, the objects are left in RECOVER pending status. You can then recover the table space using the image copy created by REORG. You must also recover the indexes. After recovery, the objects have completed reorganization.

If you terminate REORG with the TERM UTILITY command during the BUILD2 phase, the logical partition is left in RECOVER pending status. After you run REBUILD INDEX for the logical partition, all objects have completed reorganization.

If you restart a REORG job of certain catalog or directory table spaces, you cannot restart from the last checkpoint. For the following table spaces, you must specify RESTART(PHASE):

DSNDB06.SYSDBASE
DSNDB06.SYSDBAUT
DSNDB06.SYSGROUP
DSNDB06.SYSPLAN
DSNDB06.SYSVIEWS
DSNDB01.DBD01

The REORG pending status is not reset until the UTILTERM execution phase. If the REORG utility abends or is terminated, the objects are left in REORG pending and RECOVER pending status, depending on the phase where the failure occurred. See Appendix C, “Resetting an advisory or restrictive status” on page 527 for information about resetting either status.

Table 54. REORG TABLESPACE phases and restrictive statuses

Phase	Effect on restrictive status
UNLOAD	No effect
RELOAD	SHRLEVEL NONE has these effects: • 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 not specified, the COPY pending status is reset at the end of the phase. SHRLEVEL REFERENCE or CHANGE has no effect.
SORT	No effect
BUILD	SHRLEVEL NONE resets RECOVER pending status for indexes and, if both COPYDDN and SORTKEYS are specified, resets copy pending status for table spaces at the end of the phase. SHRLEVEL REFERENCE or CHANGE has no effect.
SORTBLD	Same effects as in the SORT and BUILD phases.
LOG	No effect
SWITCH	No effect. Under certain conditions, if TERM UTILITY is issued, it must complete successfully or objects may 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, which recovers the table space to its state before the failed reorganization.

Restarting REORG: Table 55 on page 328 provides information about restarting REORG TABLESPACE, depending on the phase REORG was in when the job stopped.

If you restart REORG in the UTILINIT phase, it re-executes from the beginning of the phase. If REORG abends or 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 TABLESPACE (with SHRLEVEL NONE, with SHRLEVEL REFERENCE, and with SHRLEVEL CHANGE), the table indicates the types of restart allowed (CURRENT and PHASE). "None" indicates that no restart is allowed. A blank indicates that the phase does not occur. The "Data Sets Required" column lists the data sets that must exist to perform the specified type of restart in the specified phase.

Table 55. REORG TABLESPACE utility restart information

Phase	Type for NONE	Type for REFERENCE	Type for CHANGE	Data Sets Required	Notes
UNLOAD	CURRENT PHASE	CURRENT PHASE	None None	SYSREC	
RELOAD	CURRENT PHASE	CURRENT PHASE	None None	SYSREC and SYSUT1 SYSREC	1,2 1,2
SORT	CURRENT PHASE	CURRENT PHASE	None None	SYSUT1 SYSUT1	2,3 2
BUILD	CURRENT PHASE	CURRENT PHASE	None None	SORTOUT SORTOUT	2,3,4 2,4
SORTBLD	CURRENT PHASE	CURRENT PHASE	None None		2 2
LOG			None None		
SWITCH		CURRENT PHASE	CURRENT PHASE	originals and shadows originals and shadows	3
BUILD2		CURRENT PHASE	CURRENT PHASE	shadows for nonpartitioning indexes shadows for nonpartitioning indexes	3,4 4

Note:

1. For NONE, if you specified NOSYSREC, then RESTART is not possible, and you must execute the RECOVER TABLESPACE utility for the table space or partition. For REFERENCE, if you specified both SORTDATA and NOSYSREC, then RESTART or RESTART(PHASE) restarts at the beginning of the UNLOAD phase.
2. For NONE and REFERENCE, if you specified SORTKEYS, then 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 specified 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 using RESTART CURRENT, inline statistics collection will 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 will be updated using RESTART(PHASE), see Table 56 on page 329.

Table 56. Statistics collection for REORG TABLESPACE utility phase restart

Phase	CURRENT	PHASE
UTILINIT	NO	YES
UNLOAD	NO	YES
RELOAD	NO	YES
SORT	NO	NO
BUILD	NO	YES
SORTBLD	NO	YES

For instructions on restarting a utility job, see “Chapter 2-1. Invoking DB2 online utilities” on page 27.

Restarting REORG after an out of space condition: See “Restarting after the output data set is full” on page 49 for guidance in restarting REORG from the last commit point after receiving an out of space condition.

Concurrency and compatibility

Individual data and index partitions, and individual logical partitions of nonpartitioning indexes, are treated as distinct target objects. Utilities operating 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.

REORG TABLESPACE compatibility

Table 57 on page 330, Table 58 on page 331, Table 59 on page 331, and Table 60 on page 332 show which claim classes REORG drains and any restrictive state the utility sets on the target object.

For nonpartitioning 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 respect PCTFREE or FREEPAGE attributes when inserting keys

Table 61 on page 332 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 is also shown.

Table 62 on page 333 shows which DB2 operations can be affected when reorganizing catalog table spaces.

For SHRLEVEL NONE, Table 57 on page 330 shows which claim classes REORG drains and any restrictive state 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 57. Claim classes of REORG TABLESPACE SHRLEVEL NONE operations. Use of claims and drains; restrictive states set on the target object.

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 or partition of partitioning index	DW/UTRO	DA/UTUT	DW/UTRO	DA/UTUT
Nonpartitioning index	DW/UTRO	DA/UTUT		DR
Logical partition of nonpartitioning index			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
- Blank: Any claim, drain, or restrictive state for this object does not change in this phase

For SHRLEVEL REFERENCE, Table 58 on page 331 shows which claim classes REORG drains and any restrictive state 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 58. Claim classes of REORG TABLESPACE SHRLEVEL REFERENCE operations. Use of claims and drains; restrictive states set on the target object.

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 or partition of partitioning index	DW/UTRO	DA/UTUT	DW/UTRO	DA/UTUT	UTRW
Nonpartitioning index	DW/UTRO	DA/UTUT		DR	
Logical partition of nonpartitioning index			DW/UTRO	DA/UTUT	

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
- Blank: 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 59 shows which claim classes REORG drains and any restrictive state the utility sets on the target object.

Table 59. Claim classes of REORG TABLESPACE SHRLEVEL CHANGE operations. Use of claims and drains; restrictive states set on the target object.

Target	UNLOAD phase	Last iteration of LOG phase	SWITCH phase
Table space	CR/UTRW[1]	DW/UTRO	DA/UTUT
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 NONE, Table 60 on page 332 shows which claim classes REORG drains and any restrictive state the utility sets on the target object.

Table 60. Claim classes of REORG TABLESPACE SHRLEVEL CHANGE operations. Use of claims and drains; restrictive states set on the target object.

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			DR	
Logical partition of nonpartitioning index	CR/UTRW	DW/UTRO or DA/UTUT[1]	DA/UTUT	

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
- Blank: Any claim, drain, or restrictive state for this object does not change in this phase

Notes:

1. DA/UTUT applies if you specified DRAIN ALL.

Table 61 (Page 1 of 2). REORG TABLESPACE compatibility

Action	REORG SHRLEVEL NONE UNLOAD CONTINUE or PAUSE, REORG SHRLEVEL REFERENCE, or REORG SHRLEVEL CHANGE	REORG SHRLEVEL NONE UNLOAD ONLY without cluster index	REORG SHRLEVEL NONE UNLOAD ONLY with cluster 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

Table 61 (Page 2 of 2). REORG TABLESPACE compatibility

Action	REORG SHRLEVEL NONE UNLOAD CONTINUE or PAUSE, REORG SHRLEVEL REFERENCE, or REORG SHRLEVEL CHANGE	REORG SHRLEVEL NONE UNLOAD ONLY without cluster index	REORG SHRLEVEL NONE UNLOAD ONLY with cluster index
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 TABLESPACE PAGE REPLACE	No	No	No
REPAIR LOCATE INDEX PAGE REPLACE	No	Yes	No
REPORT	Yes	Yes	Yes
RUNSTATS	No	Yes	Yes
STOSPACE	No	Yes	Yes

Table 62. DB2 operations 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	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 it consists of a reorganized index or index partition. Table 63 summarizes the effect of REORG on a table space partition and on the corresponding index partition.

Table 63. REORG summary

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 parts *n1* through *n2* + parts *n1* through *n2* of the partitioned index + index entries for those parts in all nonpartitioning indexes

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. (Those values can be set by 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 records loaded belong to the same or different tables.

Segments that contain a table that has an explicit cluster index are unloaded using the cluster index; when the table is loaded, all data records are in cluster key order.

After running REORG TABLESPACE

After a reorganization has completed:

- 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 prevents having to process the log records 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 are true:
  - It is relatively small
  - It is used only in read-only applications
  - It can be easily loaded again in the event of failure.

  See “Chapter 2-7. COPY” on page 85 for information on making image copies.
- If you use REORG SHRLEVEL NONE LOG NO on a LOB table space and the LOB manager determines that nothing needs to be done to the table space, no COPY pending status is set. However, if the LOB manager 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 can not use an image copy created before turning off REORP.

- If you use COPYDDN, SHRLEVEL REFERENCE, or SHRLEVEL CHANGE, and the object 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 only need to run RUNSTATS on nonpartitioning indexes if you reorganized a subset of the partitions.
- If you have used REORG TABLESPACE SHRLEVEL CHANGE, you can drop the mapping table and its index.
- If you have used 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 specified DISCARD on a REORG of a table involved in a referential integrity set, you will need to run CHECK DATA against any affected referentially-related objects that were placed in CHECK pending status.

Sample control statements

Example 1: REORG using default sort output data set. This example shows the DDNAME for the unload data set is UNLD, the DDNAME for the sort input data set is WORK, and the DDNAME for the sort output data set is defaulted to SORTOUT.

```
//STEP1    EXEC DSNUPROC,UID='IUJLU101.REORG',
//         UTPROC='',
//         SYSTEM='V61A'
//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)
//WORK   DD DSN=IUJLU101.REORG.STEP1.WORK,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT  DD DSN=IUJLU101.REORG.STEP1.SORTOUT,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
REORG TABLESPACE (DSN8D61A.DSN8S61D)
      SORTDATA
      UNLDDN (UNLD)
      WORKDDN (WORK)
//*
```

Example 2: Reorganizing a table space. Reorganize table space DSN8S61D in database DSN8D61A.

```
REORG TABLESPACE DSN8D61A.DSN8S61D
         SORTDATA
```

Example 3: Reorganizing a table space partition. Reorganize partition 3 of table space DSN8S61E in database DSN8D61A.

```
REORG TABLESPACE DSN8D61A.DSN8S61E
  PART 3
  SORTDATA
  SORTDEVT SYSDA
```

Example 4: REORG with DFSORT unloading by table space scan. Reorganize table space DSN8S61E in database DSN8D61A. Specify that DFSORT unloads the data by table space scan.

```
REORG TABLESPACE DSN8D61A.DSN8S61E SORTDATA
```

Example 5: REORG Using SORTKEYS. Use the SORTKEYS option to invoke parallel index build for a reorganization of the table space DSN8S61D in database DSNDB04. This example does not specify that dynamic allocation is to be used by DFSORT. Instead, it 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, which includes a DD statement that allocates UTPRINT to SYSOUT.

```
//SAMPJOB  JOB  ...
//STEP1    EXEC DSNUPROC,UID='SAMPJOB.REORG',UTPROC='',SYSTEM='V61A'
//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.DSN8S61D LOG NO SORTDATA SORTKEYS
/*
```

Example 6: REORG Using SORTKEYS while allowing read-write access. Use the SORTKEYS option to invoke parallel index build for reorganization of the table space DSN8S61E in database DSNDB04 in database DSNDB04. The name of the mapping table is DSN8MAP. DFSORT dynamically allocates sort work data sets. 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.

```
//SAMPJOB  JOB  ...
//STEP1    EXEC DSNUPROC,UID='SAMPJOB.REORG',UTPROC='',SYSTEM='V61A'
//SYSCOPY  DD UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND),
//            DSN=SAMPJOB,COPY,DISP=(NEW,CATLG,CATLG)
//SYSIN    DD *
REORG TABLESPACE DSNDB04.DSN8S61E LOG NO SORTDEVT SYSDA SORTNUM 4
  SHRLEVEL CHANGE MAPPINGTABLE DSN8MAP
/*
```

Example 7: Reorganizing a table while allowing read-only access. Reorganize table space DSN8S61D in database DSN8D61A. The deadline for start of the SWITCH phase is *3:15* on *February 4, 1997.*

```
REORG TABLESPACE DSN8D61A.DSN8S61D COPYDDN(MYCOPY1)
  RECOVERYDDN(MYCOPY2) SHRLEVEL REFERENCE
  DEADLINE 1997-2-4-03.15.00
```

Example 8: Reorganizing a table while allowing read-write access. Reorganize table space DSN8S61D in database DSN8D61A. The deadline for start of the SWITCH phase is *3:15* on February 4, 1997. The name of the mapping table is MYMAPTABLE. The maximum desired amount of time for the log processing in the read-only (last) iteration of log processing is *240* seconds. If reorganization's reading of the log is not catching up to applications' writing of the log quickly enough, DB2 will drain the write claim class after sending the LONGLOG message to the operator. That draining will take place at least 900 seconds after the LONGLOG message is sent.

```
REORG TABLESPACE DSN8D61A.DSN8S61D COPYDDN(MYCOPY1)
  RECOVERYDDN(MYCOPY2) SHRLEVEL CHANGE
  DEADLINE 1997-2-4-03.15.00
  MAPPINGTABLE
DSN8610.MAP_TBL MAXRO 240 LONGLOG DRAIN DELAY 900
```

Example 9: Reorganizing a range of table space partitions. Reorganize partitions 3 through 5 of table space DSN8S61E in database DSN8D61A.

```
REORG TABLESPACE DSN8D61A.DSN8S61E
  PART 3:5
  STATISTICS
  SORTDEVT SYSDA
    SHRLEVEL NONE
    COPYDDN SYSCOPY
    SORTDATA
```

Example 10: REORG a partition using STATISTICS. Reorganize partition 3 of table space DSN8S61E in database DSN8D61A, using the STATISTICS option to update catalog table statistics for that table.

```
REORG TABLESPACE DSN8D61A.DSN8S61E
   SORTDATA STATISTICS PART 3
```

Example 11: REORG using STATISTICS to update table space and index statistics. Reorganize table space DSN8S61E in database DSN8D61A, using the STATISTICS option to update catalog statistics for the table space and all indexes defined on that table.

```
REORG TABLESPACE DSN8D61A.DSN8S61E SORTDATA STATISTICS
   TABLE
   INDEX(ALL) KEYCARD FREQVAL NUMCOLS 1
   COUNT 10 REPORT YES UPDATE NONE
```

Example 12: Checking if a table should be reorganized. Report if the OFFPOSLIMIT or INDREFLIMIT values are exceeded for the TPHR5201 table space in database DBHR5201.

```
//STEP1    EXEC DSNUPROC,UID='HUHRU252.REORG2',TIME=1440,
//         UTPROC='',
//         SYSTEM='V61A',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)
//SYSUT1   DD DSN=HUHRU252.REORG2.STEP1.SYSUT1,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT  DD DSN=HUHRU252.REORG2.STEP1.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
REORG TABLESPACE DBHR5201.TPHR5201 PART 11
      SORTDATA SORTKEYS NOSYSREC
      REPORTONLY
      SHRLEVEL CHANGE MAPPINGTABLE ADMF001.MAP1
      COPYDDN (SYSCOPY)
          OFFPOSLIMIT 11 INDREFLIMIT 15
          STATISTICS UPDATE SPACE
/*
```

On successful completion, DB2 returns output similar to the following output:

```
DSNU050I   DSNUGUTC - REORG TABLESPACE DBHR5201.TPHR5201 PART 11 SORTDATA SORTKEYS NOSYSREC REPORTONLY SHRLEVEL CHA
NGE MAPPINGTABLE ADMF001.MAP1 COPYDDN(SYSCOPY) OFFPOSLIMIT 11 INDREFLIMIT 15 STATISTICS UPDATE SPACE
DSNU286I ( DSNURLIM - REORG TABLESPACE DBHR5201.TPHR5201 OFFPOSLIMIT SYSINDEXPART ROWS
  CREATOR .IXNAME             CREATOR .TBNAME             PART CARDF       FAROFFPOSF  NEAROFFPOSF      STATSTIME
* ADMF001 .IPHR5201           ADMF001 .TBHR5201             11  6.758E+03   2.972E+03   7.38E+02   1999-02-05-08.27.04
DSNU287I ( DSNURLIM - REORG TABLESPACE DBHR5201.TPHR5201 INDREFLIMIT SYSTABLEPART ROWS
  DBNAME  .TSNAME   PART         CARD  FARINDREF NEARINDREF      STATSTIME
  DBHR5201.TPHR5201   11         6758          0          0 1999-02-05-08.27.04
DSNU289I ( DSNURLIM -  REORG LIMITS HAVE BEEN MET
DSNU010I   DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=2
DSNU000I   DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = HUHRU252.REORG2
DSNU050I   DSNUGUTC - REORG INDEX ADMF001.IPHR5201 PART 11 LEAFDISTLIMIT 2 REPORTONLY
DSNU288I ( DSNURLIM - REORG INDEX ADMF001.IPHR5201 LEAFDISTLIMIT SYSINDEXPART ROWS
  CREATOR .IXNAME             PART   LEAFDIST      STATSTIME
* ADMF001 .IPHR5201             11          3 1999-02-05-08.27.04
DSNU289I ( DSNURLIM -  REORG LIMITS HAVE BEEN MET
DSNU050I   DSNUGUTC - CHECK INDEX(ADMF001.IPHR5201 PART 11)
DSNU700I ( DSNUKGET - 6761 INDEX ENTRIES UNLOADED FROM INDEX='ADMF001.IPHR5201' PARTITION= 11
DSNU705I   DSNUK001 - UNLOAD PHASE COMPLETE - ELAPSED TIME=00:00:00
DSNU042I   DSNUGSOR - SORT PHASE STATISTICS -
                      NUMBER OF RECORDS=6761
                      ELAPSED TIME=00:00:01
DSNU717I ( DSNUKTER - 6761 ENTRIES CHECKED FOR INDEX 'ADMF001.IPHR5201' PARTITION= 11
DSNU720I   DSNUK001 - CHECKIDX PHASE COMPLETE, ELAPSED TIME=00:00:02
DSNU010I   DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=2
```

Figure 19. Sample output showing REORG limits have been met

Example 13: Conditionally reorganizing a table. To ensure recent statistics for the table space, execute the RUNSTATS utility for the TPHR5201 table space.

Then, reorganize the TPHR5201 table space in database DBHR5201 if the OFFPOSLIMIT or INDREFLIMIT value is exceeded.

```
//*******************************************************************
//* COMMENT: UPDATE STATISTICS
//*******************************************************************
//STEP1    EXEC DSNUPROC,UID='HUHRU252.REORG1',TIME=1440,
//         UTPROC='',
//         SYSTEM=V61AR',DB2LEV=DB2A
//SYSREC   DD DSN=HUHRU252.REORG1.STEP1.SYSREC,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSUT1   DD DSN=HUHRU252.REORG1.STEP1.SYSUT1,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT  DD DSN=HUHRU252.REORG1.STEP1.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         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='V61A',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)
//SYSUT1   DD DSN=HUHRU252.REORG1.STEP1.SYSUT1,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT  DD DSN=HUHRU252.REORG1.STEP1.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//         SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
REORG TABLESPACE DBHR5201.TPHR5201
      SORTDATA NOSYSREC SORTKEYS
      COPYDDN SYSCOPY1
      OFFPOSLIMIT
      INDREFLIMIT
      STATISTICS TABLE(ALL) INDEX(ALL)
/*
```

On successful completion, DB2 returns output similar to the following output:

```
DSNU000I   DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = HUHRU252.REORG1
DSNU050I   DSNUGUTC - RUNSTATS TABLESPACE DBHR5201.TPHR5201 UPDATE SPACE
DSNU610I - DSNUSUTP -   SYSTABLEPART CATALOG UPDATE FOR DBHR5201.TPHR5201 SUCCESSFUL
DSNU620I - DSNUSDRA - RUNSTATS CATALOG TIMESTAMP = 1999-05-03-14.57.58.921242
DSNU010I   DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
DSNU000I   DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = HUHRU252.REORG1
DSNU050I   DSNUGUTC - REORG TABLESPACE DBHR5201.TPHR5201 SORTDATA NOSYSREC SORTKEYS COPYDDN(SYSCOPY1) OFFPOSLIMIT
INDREFLIMIT STATISTICS TABLE ALL INDEX(ALL)
DSNU286I - DSNURLIM - REORG TABLESPACE DBHR5201.TPHR5201 OFFPOSLIMIT SYSINDEXPART ROWS
  CREATOR .IXNAME             CREATOR .TBNAME             PART CARDF       FAROFFPOSF  NEAROFFPOSF      STATSTIME
* ADMF001 .IPHR5201           ADMF001 .TBHR5201              1  3.6E+01     8.0E+00     4.0E+00    1999-05-03-14.57.43
  ADMF001 .IPHR5201           ADMF001 .TBHR5201              2  5.0E+00     0.0E0       0.0E0      1999-05-03-14.57.43
  ADMF001 .IPHR5201           ADMF001 .TBHR5201              3  5.4E+01     0.0E0       0.0E0      1999-05-03-14.57.43
  ADMF001 .IPHR5201           ADMF001 .TBHR5201              4  3.0E+01     1.0E+00     0.0E0      1999-05-03-14.57.43
* ADMF001 .IPHR5201           ADMF001 .TBHR5201              5  2.1E+01     2.0E+00     1.0E+00    1999-05-03-14.57.43
  ADMF001 .IPHR5201           ADMF001 .TBHR5201              6  5.0E+00     0.0E0       0.0E0      1999-05-03-14.57.43
  ADMF001 .IPHR5201           ADMF001 .TBHR5201              7  4.0E+00     0.0E0       0.0E0      1999-05-03-14.57.43
* ADMF001 .IPHR5201           ADMF001 .TBHR5201              8  3.5E+01     9.0E+00     8.0E+00    1999-05-03-14.57.43
* ADMF001 .IPHR5201           ADMF001 .TBHR5201              9  2.5E+01     4.0E+00     1.0E+00    1999-05-03-14.57.43
  ADMF001 .IPHR5201           ADMF001 .TBHR5201             10  1.0E+00     0.0E0       0.0E0      1999-05-03-14.57.43
* ADMF001 .IPHR5201           ADMF001 .TBHR5201             11  6.758E+03   2.972E+03   7.38E+02   1999-05-03-14.57.43
DSNU287I - DSNURLIM - REORG TABLESPACE DBHR5201.TPHR5201 INDREFLIMIT SYSTABLEPART ROWS
  DBNAME  .TSNAME   PART         CARD  FARINDREF NEARINDREF      STATSTIME
  DBHR5201.TPHR5201    1           36          0          0 1999-05-03-14.57.58
  DBHR5201.TPHR5201    2            5          0          0 1999-05-03-14.57.58
  DBHR5201.TPHR5201    3           54          0          0 1999-05-03-14.57.58
  DBHR5201.TPHR5201    4           30          0          0 1999-05-03-14.57.58
  DBHR5201.TPHR5201    5           21          0          0 1999-05-03-14.57.58
  DBHR5201.TPHR5201    6            5          0          0 1999-05-03-14.57.58
  DBHR5201.TPHR5201    7            4          0          0 1999-05-03-14.57.58
  DBHR5201.TPHR5201    8           35          0          0 1999-05-03-14.57.58
  DBHR5201.TPHR5201    9           25          0          0 1999-05-03-14.57.58
  DBHR5201.TPHR5201   10            1          0          0 1999-05-03-14.57.58
  DBHR5201.TPHR5201   11         6758          0          0 1999-05-03-14.57.58
DSNU289I - DSNURLIM -  REORG LIMITS HAVE BEEN MET
DSNU290I - DSNURLIM -  REORG WILL BE PERFORMED
DSNU252I   DSNUGSRT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=6974 FOR TABLESPACE DBHR5201.TPHR5201
DSNU250I   DSNUGSRT - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:03
DSNU400I   DSNURBID - COPY PROCESSED FOR TABLESPACE DBHR5201.TPHR5201
                      NUMBER OF PAGES=1034
                      AVERAGE PERCENT FREE SPACE PER PAGE = 14.96
                      PERCENT OF CHANGED PAGES =100.00
                      ELAPSED TIME=00:01:46
DSNU610I - DSNUSUTP -   SYSTABLEPART CATALOG UPDATE FOR DBHR5201.TPHR5201 SUCCESSFUL
DSNU610I - DSNUSUPT -       SYSTABSTATS CATALOG UPDATE FOR ADMF001.TBHR5201 SUCCESSFUL
DSNU610I - DSNUSUPC -         SYSCOLSTATS CATALOG UPDATE FOR ADMF001.TBHR5201 SUCCESSFUL
DSNU610I - DSNUSUTB -     SYSTABLES CATALOG UPDATE FOR ADMF001.TBHR5201 SUCCESSFUL
DSNU610I - DSNUSUCO -       SYSCOLUMNS CATALOG UPDATE FOR ADMF001.TBHR5201 SUCCESSFUL
DSNU610I - DSNUSUTS -   SYSTABLESPACE CATALOG UPDATE FOR DBHR5201.TPHR5201 SUCCESSFUL
DSNU620I - DSNURDRT - RUNSTATS CATALOG TIMESTAMP = 1999-05-03-14.58.16.924784
DSNU304I - DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=6974 FOR TABLE ADMF001.TBHR5201
DSNU302I   DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=6974
DSNU300I   DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:01:48
DSNU042I   DSNUGSOR - SORT PHASE STATISTICS -
                      NUMBER OF RECORDS=27896
                      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
DSNU348I - DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=54 FOR INDEX ADMF001.IPHR5201  PART 3
DSNU348I - DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=30 FOR INDEX ADMF001.IPHR5201  PART 4
DSNU348I - DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=21 FOR INDEX ADMF001.IPHR5201  PART 5
DSNU348I - DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=5 FOR INDEX ADMF001.IPHR5201  PART 6
DSNU348I - DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=4 FOR INDEX ADMF001.IPHR5201  PART 7
DSNU348I - DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=35 FOR INDEX ADMF001.IPHR5201  PART 8
```

Figure 20 (Part 1 of 2). Sample showing RUNSTATS and conditional REORG output

```
DSNU348I - DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=25 FOR INDEX ADMF001.IPHR5201  PART 9
DSNU348I - DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=1 FOR INDEX ADMF001.IPHR5201  PART 10
DSNU348I - DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=6758 FOR INDEX ADMF001.IPHR5201  PART 11
DSNU349I - DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=6974 FOR INDEX ADMF001.IXHR521B
DSNU349I - DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=6974 FOR INDEX ADMF001.IXHR521C
DSNU349I - DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=6974 FOR INDEX ADMF001.IXHR521D
DSNU610I - DSNUSUIP -   SYSINDEXPART CATALOG UPDATE FOR ADMF001.IPHR5201 SUCCESSFUL
DSNU610I - DSNUSUPI -       SYSINDEXSTATS CATALOG UPDATE FOR ADMF001.IPHR5201 SUCCESSFUL
DSNU610I - DSNUSUPD -         SYSCOLDISTSTATS CATALOG UPDATE FOR ADMF001.IPHR5201 SUCCESSFUL
DSNU610I - DSNUSUPC -         SYSCOLSTATS CATALOG UPDATE FOR ADMF001.TBHR5201 SUCCESSFUL
DSNU610I - DSNUSUIX -     SYSINDEXES CATALOG UPDATE FOR ADMF001.IPHR5201 SUCCESSFUL
DSNU610I - DSNUSUCO -       SYSCOLUMNS CATALOG UPDATE FOR ADMF001.TBHR5201 SUCCESSFUL
DSNU610I - DSNUSUCD -       SYSCOLDIST CATALOG UPDATE FOR ADMF001.IPHR5201 SUCCESSFUL
DSNU610I - DSNUSUIP -   SYSINDEXPART CATALOG UPDATE FOR ADMF001.IXHR521B SUCCESSFUL
DSNU610I - DSNUSUIX -     SYSINDEXES CATALOG UPDATE FOR ADMF001.IXHR521B SUCCESSFUL
DSNU610I - DSNUSUCO -       SYSCOLUMNS CATALOG UPDATE FOR ADMF001.TBHR5201 SUCCESSFUL
DSNU610I - DSNUSUCD -       SYSCOLDIST CATALOG UPDATE FOR ADMF001.IXHR521B SUCCESSFUL
DSNU610I - DSNUSUIP -   SYSINDEXPART CATALOG UPDATE FOR ADMF001.IXHR521C SUCCESSFUL
DSNU610I - DSNUSUIX -     SYSINDEXES CATALOG UPDATE FOR ADMF001.IXHR521C SUCCESSFUL
DSNU610I - DSNUSUCO -       SYSCOLUMNS CATALOG UPDATE FOR ADMF001.TBHR5201 SUCCESSFUL
DSNU610I - DSNUSUCD -       SYSCOLDIST CATALOG UPDATE FOR ADMF001.IXHR521C SUCCESSFUL
DSNU610I - DSNUSUIP -   SYSINDEXPART CATALOG UPDATE FOR ADMF001.IXHR521D SUCCESSFUL
DSNU610I - DSNUSUIX -     SYSINDEXES CATALOG UPDATE FOR ADMF001.IXHR521D SUCCESSFUL
DSNU610I - DSNUSUCO -       SYSCOLUMNS CATALOG UPDATE FOR ADMF001.TBHR5201 SUCCESSFUL
DSNU610I - DSNUSUCD -       SYSCOLDIST CATALOG UPDATE FOR ADMF001.IXHR521D SUCCESSFUL
DSNU620I - DSNURDRI - RUNSTATS CATALOG TIMESTAMP = 1999-05-03-15.00.05.010574
DSNU258I   DSNURBXD - BUILD PHASE STATISTICS - NUMBER OF INDEXES=4
DSNU259I   DSNURBXD - BUILD PHASE COMPLETE, ELAPSED TIME=00:00:11
DSNU405I   DSNURORG - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DBHR5201.TPHR5201
DSNU010I   DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=2
```

Figure 20 (Part 2 of 2). Sample showing RUNSTATS and conditional REORG output

Chapter 2-17. REPAIR

The REPAIR online utility repairs data. The data can be your own data, or data you would not normally access, such as space map pages and index entries.

REPAIR is intended as a means of replacing invalid data with valid data. Be extremely careful using REPAIR. Improper use can damage the data even further.

You can use the REPAIR utility to:

- Test 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 OBDs for a LOB table space

For a diagram of REPAIR syntax and a description of available options, see “Syntax and options of the control statement” on page 344. For detailed guidance on running this utility, see “Instructions for running REPAIR” on page 355.

Output: The potential output from the REPAIR utility consists of a modified page or pages in the specified DB2 table space or index, and a dump of the contents.

Authorization required: To execute this utility, the privilege set of the process must include one of the following:

- 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 DBD, the privilege set must include SYSADM, SYSCTRL, or installation SYSOPR authority.

REPAIR should only be used by a knowledgeable person. Be careful to grant REPAIR authorization only to appropriate people.

Execution phases of REPAIR: One of the following phases can be identified if the job terminates.

The phases for REPAIR are:

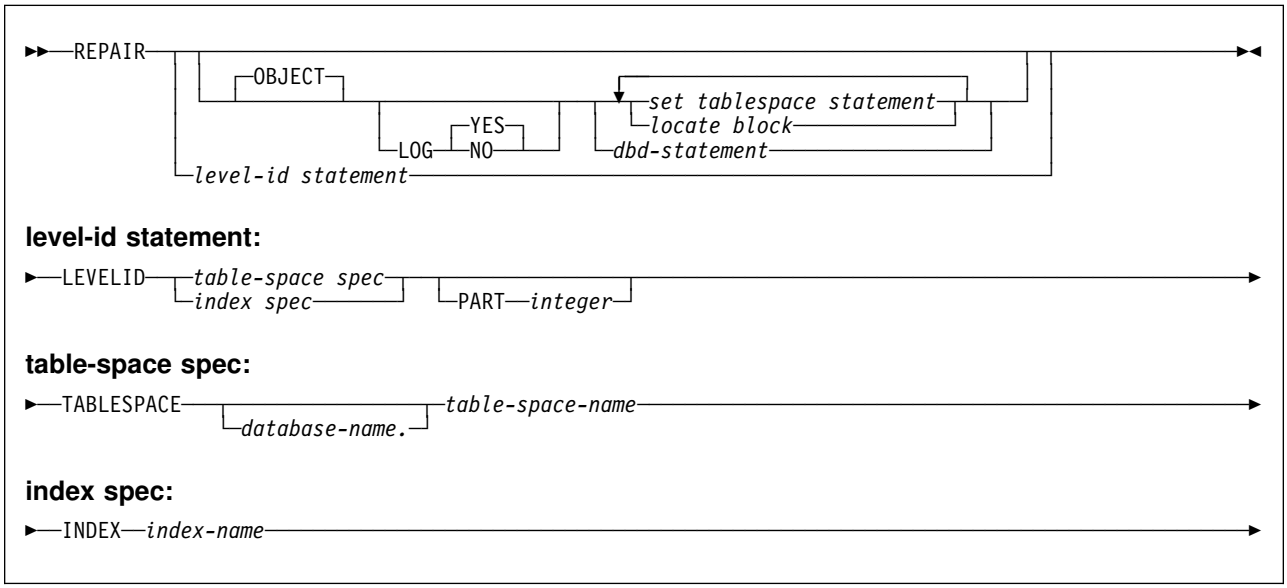
Phase	Description
UTILINIT	Initialization
REPAIR	
UTILTERM	Cleanup.

Syntax and options of the control statement

The utility control statement defines the function 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

For guidance in interpreting syntax diagrams, see "How to read the syntax diagrams" on page 4.



REPAIR option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see "Control statement coding rules" on page 27.

OBJECT Is optional, and used for clarity only.

LOG Tells whether to log the changes made by REPAIR. If the changes are logged, they are applied again if the data is recovered.

YES Logs the changes.

The **default** is **LOG YES**.

REPAIR LOG YES cannot override the LOG NO attribute of a table space.

NO Does not log the changes. You cannot use this option with a DELETE statement.

REPAIR LOG NO can override the LOG YES attribute of a table space.

LEVELID Sets the level identifier of the named table space, table space partition, index, or index space partition to a new identifier. Use LEVELID to accept the use of a down-level data set. You cannot specify multiple LEVELIDs.

You cannot use LEVELID with a table space, table space partition, index, or index space partition with 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 resulting 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 set.

database-name Specifies the name of the database the table space belongs to.

The **default** is **DSNDB04**.

table-space-name Specifies the name of the table space.

INDEX Specifies the index whose level identifier is set.

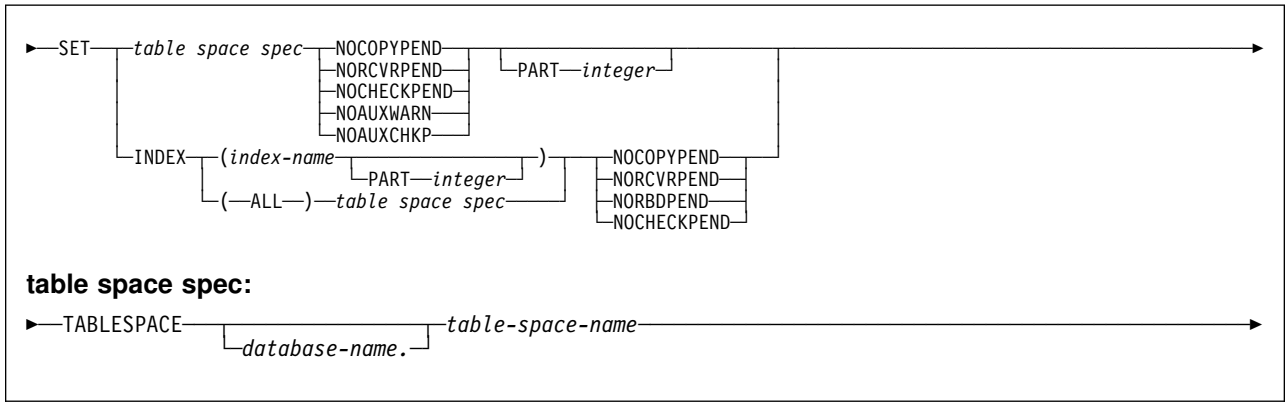
index-name Specifies the index to be processed.

PART Identifies a partition of the table space or index.

integer is the number of the partition and must be in the range from one to the number of partitions defined for the object. The maximum is 254.

SET TABLESPACE and SET INDEX 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.



SET TABLESPACE and SET INDEX option descriptions

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 the table space belongs to.

The **default** is **DSNDB04**.

table-space-name Specifies the name of the table space.

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 to be processed.

(ALL) Specifies that all indexes in the table space will be processed.

The keyword INDEXES is still accepted following a *table space spec* and causes all indexes to be processed. All indexes can also be processed by specifying INDEX(ALL) followed by a *table space spec*.

NOCOPYPEND Resets the COPY pending status of the specified table space, or informational COPY pending (ICOPY) status of the specified index.

NORCVRPEND Resets the RECOVER pending (RECP) status of the specified table space or index.

NORBDPEND Resets the REBUILD pending (RBDP) status, the page set REBUILD pending status (PSRBDP), or the RBDP* status of the specified index.

NOCHECKPEND Resets the CHECK pending (CHECKP) status of the specified table space or index.

NOAUXWARN Resets the auxiliary warning (AUXW) status of the specified table space. The specified table space must be a base table space or a LOB table space.

NOAUXCHKP Resets the auxiliary CHECK pending (ACHKP) status of the specified table space. The specified table space must be a base table space.

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 defined for the object. The maximum is 254.

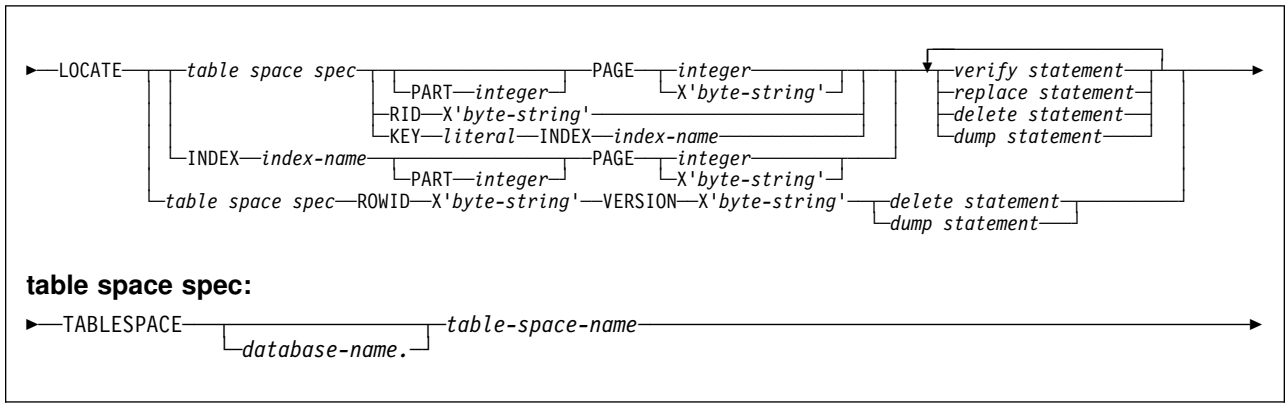
You can specify PART for NOCHECKPEND on a table space, and for NORCVRPEND on indexes.

The PART keyword is not valid for a LOB table space or an index on the auxiliary table.

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. There can be 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.



LOCATE TABLESPACE statement option descriptions

The LOCATE TABLESPACE statement locates data to be repaired within a table space.

One LOCATE statement is required for each unit of data to be repaired. Several LOCATE statements can appear after each REPAIR statement.

If a REPAIR statement is followed by several LOCATE statements, all processing caused by VERIFY, REPLACE, and DUMP statements is committed before the next LOCATE 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 containing the data you want to repair.

PART *integer* Valid only for partitioned table spaces.

integer is the number of the partition containing the page to be located.

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 offsets given in *byte-string* and in subsequent statements are relative to the beginning of the page. The first byte of the page is 0.

byte-string is a value from one to eight hexadecimal 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 to be located is a single row. The offsets given in *byte-string* and in subsequent statements are relative to the beginning of the row. The first byte of the stored row prefix is 0.

byte-string can be one to eight hexadecimal characters. You do not need to enter leading zeros. Enclose the byte string between single quotation marks and precede it with an X.

KEY *literal* Specifies that the data to be located is a single row, identified by *literal*. The offsets given 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 specified within the LOCATE KEY option may not be specified as ASCII character strings. No conversion of the values is performed. To use this option when the table space is ASCII, the values should be specified 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.

INDEX *index-name*
Specifies a particular index that is to be used to find the row containing the key. When you are locating by key, the index you specify must be single-column.

index-name is the qualified or unqualified name of the index.

ROWID X'*byte-string***'**
Specifies that the data to be located is a LOB in a LOB table space.

byte-string is the row ID that identifies the LOB column.

The most likely source of a row ID is an orphaned LOB that is reported by 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 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.

The most likely source of a version number is an orphaned LOB that is reported by 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 option descriptions

The LOCATE INDEX statement locates data to be repaired within an index.

One LOCATE statement is required for each unit of data to be repaired. Several LOCATE statements can appear after each REPAIR statement.

If a REPAIR statement is followed by several LOCATE statements, all processing caused by VERIFY, REPLACE, and DUMP statements is committed before the next LOCATE is processed.

index-name Specifies the index containing the data you want to repair.

index-name is the qualified name of the index, in the form *creator-id.index-name*. If you omit the qualifier creator ID, the user identifier for the utility job is used.

PART *integer* Valid only for indexes of partitioned table spaces.

integer is the number of the partitioning index containing the page to be located.

PAGE *integer* Specifies the relative page number within the index space to be operated on. The first page is 0 (zero).

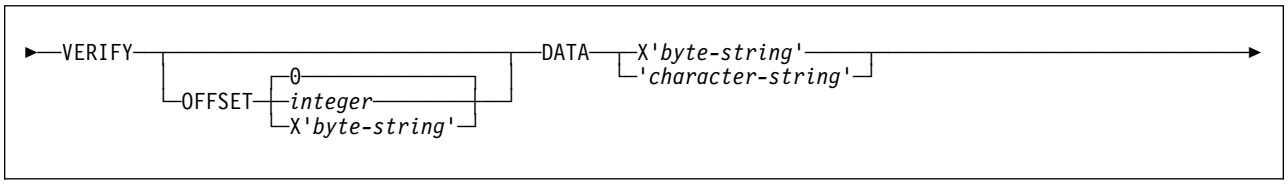
integer *integer* is a decimal number from one to six digits in length.

X'*byte-string***'** Specifies that the data of interest is an entire page. The offsets given in *byte-string* and in subsequent statements are relative to the beginning of the page. The first byte of the page is 0.

byte-string is a value from one to eight hexadecimal 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. 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 to be tested by a relative byte address within the row or page.

integer Gives the offset as an integer. The **default** is **0**, the first byte of the area identified by the previous LOCATE statement.

X'*byte-string***'** Gives the offset as one to four hexadecimal characters. You need not enter leading zeros. Enclose the byte string between single quotation marks and precede it with X.

DATA Tells what data is assumed to be present before a change is made.

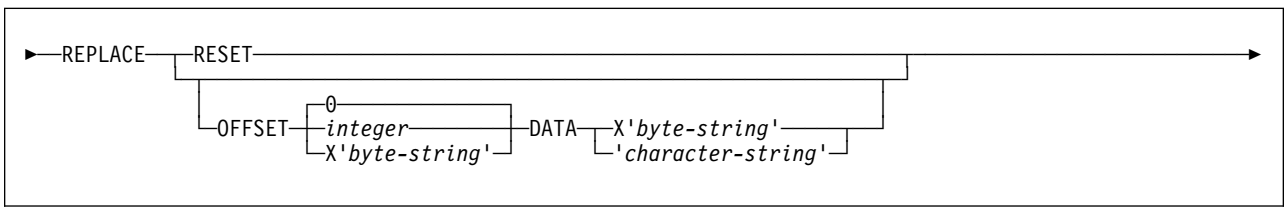
Character constants specified within the VERIFY DATA option may not be specified as ASCII character strings. No conversion of the values is performed. To use this option when the table space is ASCII, the values should be specified as hexadecimal constants.

X'*byte-string***'** Can be an *even number*, from two to thirty-two, of hexadecimal characters that must be present. You need not enter leading zeros. Enclose the byte string between single quotation marks and precede it with X.

'*character-string***'** Can be any character string that must be present.

REPLACE statement syntax

The REPLACE statement replaces data at a particular location. The statement falls 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 Resets the inconsistent data indicator. 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 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 Tells where data is to be replaced by a relative byte address within the row or page.

integer Gives the offset as an integer. The **default** is **0**, the first byte of the area identified by the previous LOCATE statement.

X'*byte-string***'** Gives the offset as one to four hexadecimal characters. You need not enter leading zeros. Enclose the byte string between single quotation marks and precede it with X.

DATA Defines the new data that is to be entered. Only one OFFSET and one DATA specification are acted on for each REPLACE statement.

Character constants specified within the VERIFY DATA option may not be specified as ASCII character strings. No conversion of the values is performed. To use this option when the table space is ASCII, the values should be specified as hexadecimal constants.

X'*byte-string***'** Can be an *even number*, from two to thirty-two, of hexadecimal characters to replace the current data. You need not enter leading zeros. Enclose the byte string between single quotation marks and precede it with X.

'*character-string***'** Can be any character string 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 falls 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.

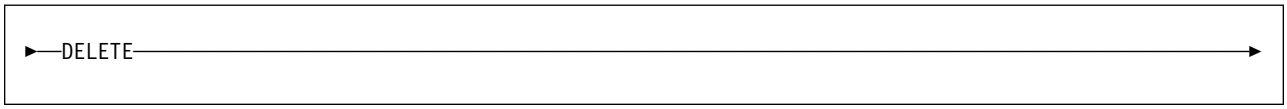
The DELETE statement operates without regard for referential constraints. If you delete a parent row, its dependent rows remain unchanged in the table space. However, in the DB2 catalog and directory table spaces, where links are used to reference other tables in the catalog, deleting a parent row causes 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 page 363 for a possible method of deleting a child row without updating its predecessor and successor.

In any LOCATE block, you can use DELETE only once.

You cannot use DELETE if you have used any of these options:

- 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.

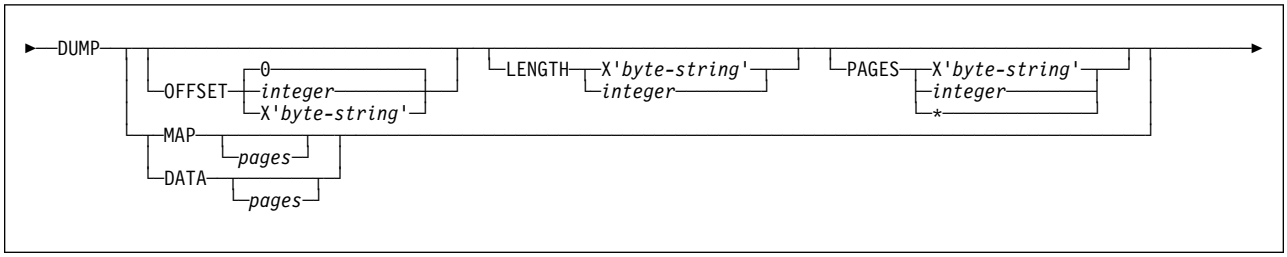
When you specify LOCATE ROWID for a LOB table space, the LOB specified by ROWID is deleted with its index entry. All pages occupied by the LOB are converted to free space. The DELETE statement will *not* remove any reference to the deleted LOB from the base table space.



DUMP statement syntax

The DUMP statement produces a hexadecimal dump of data 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 to be dumped by a relative byte address within the row or page.

integer Gives the offset as an integer. The **default** is **0**, the first byte of the row or page.

X'*byte-string***'** Gives the offset as one to four hexadecimal characters. You need not enter leading zeros. Enclose the byte string between single quotation marks and precede it with X.

LENGTH Optionally, tells the number of bytes of data to dump. If you omit both LENGTH and PAGE, the dump continues from OFFSET to the end of the row or page.

If you give 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, at the same offset.

X'*byte-string***'** Can be one to four hexadecimal characters. You need not enter leading zeros. Enclose the byte string between single quotation marks and precede it with X.

integer Gives the number as an integer.

PAGES Optionally, tells a number of pages to dump. You can use this option only if you used PAGE in the preceding LOCATE TABLESPACE control statement.

X'*byte-string***'** Can be one to four hexadecimal characters. You need not enter leading zeros. Enclose the byte string between single quotation marks and precede it with X.

integer Gives the number as an integer.

***** Dumps all pages from the starting point to the end of the table space or partition.

| **MAP** *pages* Dumps only the LOB map pages.

| *pages* specifies the number of LOB map pages to dump. If you do not specify *pages*, all LOB map pages of the LOB that is specified by ROWID and version are dumped.

| **DATA** *pages* Dumps only the LOB data pages.

| *pages* specifies the number of LOB data pages to dump. 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

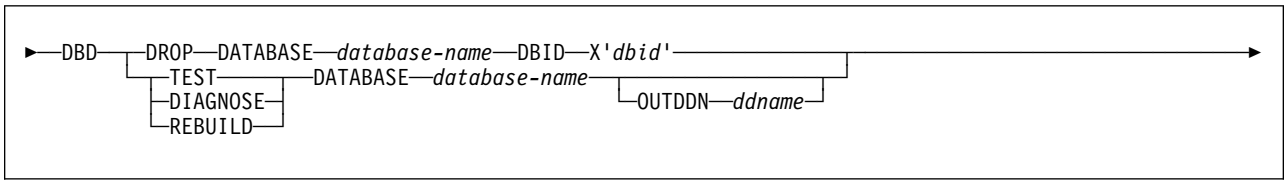
The DBD statement allows you to:

- Compare the definition of a database in the DB2 catalog with its definition in the DB2 directory
- Rebuild a database definition in the directory from the information in the DB2 catalog, including LOB information
- Drop an inconsistent database definition from the DB2 catalog and the DB2 directory.

REPAIR also assumes that the links in table spaces DSNDB01.DBD01, DSNDB06.SYSDBAUT, DSNDB06.SYSDBASE are intact. Before executing REPAIR with the DBD statement, run the DSN1CHKR utility (page 439) on these table spaces to ensure that the links are not broken.

The database on which REPAIR DBD is run must be started for access by utilities only. For more information about using the DBD statement, see page “Using the DBD statement” on page 358.

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 Drops the specified database from both the DB2 catalog and the DB2 directory. 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.

Attention: Use the DROP option with extreme care. For further assistance, you can contact the IBM Support Center.

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 Builds a DBD from information in the DB2 catalog, and compares it with the DBD in the DB2 directory. TEST reports significant differences between the two DBDs.

If the condition code is 0, then 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 can be inconsistent. Run REPAIR DBD with the DIAGNOSE option to gather information necessary for resolving any possible inconsistency.

DIAGNOSE Produces information necessary for resolving an inconsistent database definition. Like the TEST option, DIAGNOSE builds a DBD based on the information in the DB2 catalog and compares it with the DBD in the DB2 directory. In addition, DIAGNOSE reports any differences between the two DBDs, and produces hexadecimal dumps of the inconsistent DBDs.

If the condition code is 0, then the information in the DB2 catalog and the DBD in the DB2 directory is consistent.

If the condition code is 8, then the information in the DB2 catalog and the DBD in the DB2 directory can be inconsistent.

Use your electronic link with IBM Support Center, if available, for help in resolving any inconsistencies.

REBUILD Rebuilds the DBD associated with the specified database from the information in the DB2 catalog.

Attention: Use the REBUILD option with extreme care. For further assistance, you can contact the IBM Support Center.

OUTDDN *ddname*
Specifies the DD statement for an optional output data set. This data set contains copies of the DB2 catalog records 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 chapter.
2. Prepare the necessary data sets, as described in 343.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27. (For examples of JCL for REPAIR, see “Sample control statements” on page 363.)
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 357. (For a complete description of the syntax and options for REPAIR, see “Syntax and options of the control statement” on page 344.)
5. Check the compatibility table in “Concurrency and compatibility” on page 359 if you want to run other jobs concurrently on the same target objects.
6. Plan for restart if the REPAIR job doesn't complete, as described in “Terminating or restarting REPAIR” on page 359.
7. Run REPAIR.

Attention: Be extremely careful when using the REPAIR utility to replace data. Changing data to invalid values 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.

See “Chapter 2-1. Invoking DB2 online utilities” on page 27 for an explanation of ways to execute DB2 utilities.

Before running REPAIR

Making a copy of the table space

Before starting to use REPAIR to change data, it may be useful to have a copy (full image copy or DSN1COPY) of the affected table space to enable fallback.

Restoring damaged indexes

Because REPAIR can only access index data by referring to a page and an offset within the page, identifying a problem and correcting it can be difficult. Use REBUILD INDEX or RECOVER INDEX to restore damaged index data.

Data sets used by REPAIR

Table 64 describes the data sets used by REPAIR. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 64. Data sets used by REPAIR

Data Set	Description	Required?
SYSIN	Input data set containing the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
Optional output data set	Data set containing copies of the DB2 catalog records use to rebuild the DBD. The DD name is defined by the user.	No

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

Table space
Object to be repaired. It is named in the REPAIR control statement and is accessed through the DB2 catalog.

Calculating output data set size: Use the following calculation to estimate the size of the output data set:

SPACE = (4096,(n,n))

where *n* = total number of records in your catalog relating to the database on which REPAIR DBD is being executed.

An estimate for *n* can be calculated by summing the results of SELECT COUNT(*) from all of the catalog tables in the SYSDBASE table space where the name of the database associated with the record matches the database on which REPAIR DBD is being executed.

Creating the control statement

See “Syntax and options of the control statement” on page 344 for REPAIR syntax and option descriptions. See “Sample control statements” on page 363 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, it is better to reset the COPY pending restriction by taking a full image copy rather than using REPAIR. This is because RECOVER can not be executed successfully until an image copy has been made.

It is better to reset the RECOVER pending status by running RECOVER or LOAD rather than using REPAIR. This is because RECOVER uses DB2 controlled recovery information, while REPAIR SET TABLESPACE or INDEX resets the recovery pending status without considering the recoverability of the table space, such as the availability of image copies, of rows in SYSIBM.SYSCOPY, and of log data sets.

It is better to verify and possibly correct referential integrity constraints by running CHECK DATA. CHECK DATA performs a complete check of all referential integrity constraints of the table space set, while REPAIR leaves you with the responsibility of checking all the referential integrity constraints violations.

| To reset the CHECK pending status for a LOB table space, it is recommended that you:

| 1. Run the CHECK DATA utility again with the AUXERROR INVALIDATE keywords specified, then
| 2. Update the invalid LOBs.

| To reset the auxiliary warning (AUXW) status for a LOB table space, it is recommended that you:

| 1. Update the invalid LOBs, then
| 2. Run the CHECK LOB utility.

Resetting index space status

| It is better to run COPY INDEXSPACE to reset the informational COPY pending status than to use 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. This is because RECOVER uses DB2 controlled recovery information, while REPAIR SET INDEX resets the rebuild pending status without considering the recoverability of the index, such as 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 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. Keep track of your actions in case you need to undo anything later.
3. If you determined 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

The following is the recommended procedure for using the DBD statement:

1. Run the DSN1CHKR utility on the DSNDB01.DBD01, DSNDB06.SYSDBAUT, and DSNDB06.SYSDBASE table spaces.
2. Issue the STOP DATABASE (*database-name*) command, then issue the START DATABASE (*database-name*) ACCESS(UT) command to allow only utilities to access the database that is associated with the DBD.
3. 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.
4. If inconsistencies exist (condition code is not 0), use the DIAGNOSE option with the OUTDDN keyword to produce diagnostic information. Use your electronic link with IBM Support Center, if available, for assistance in analyzing this information.
5. You might be instructed to 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 will be incorrect.

   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, then the REBUILD abends.
6. If you think there is 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 your IBM Support Center for assistance.

Locating rows by key

If you use LOCATE TABLESPACE KEY, a number of rows might satisfy the condition. In this case, REPAIR only returns the RIDs of the rows, and does not perform any VERIFY, REPLACE, DELETE or DUMP which might be coded in that LOCATE block. The RID option of LOCATE TABLESPACE can then be used to identify a specific row. Examples of the messages issued are shown below:

```
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 only supports single column indexes. The following message will be received if an attempt is made to locate a row using a multiple column index.

```
DSNUCBRK - INDEX USED HAS MULTIPLE-FIELD KEY
```

Using VERIFY, REPLACE, and DELETE operations

If *any* data area *does not* contain the value required by a VERIFY statement, *all* REPLACE and DELETE operations in the same locate block are inhibited. VERIFY and REPLACE statements following the next LOCATE are not affected.

Repairing critical catalog table spaces and indexes

An ID with a granted authority receives message DSNT500I, “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 using an authorization ID with the installation SYSADM or installation SYSOPR authority.

Terminating or restarting REPAIR

REPAIR can be terminated 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 DSNU181I, and the job abends. You must terminate the job with the TERM UTILITY command, and rerun REPAIR from the beginning.

Concurrency and compatibility

Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions of the same table space or index space are compatible.

Table 65 on page 360 shows which claim classes REPAIR drains and any restrictive state the utility sets on the target object.

Table 66 on page 360 and Table 67 on page 361 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 is also shown.

Table 65. Claim classes of REPAIR operations. Use of claims and drains; restrictive states set on the target object.

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	
REPAIR LOCATE RID DELETE	DA/UTUT	DA/UTUT
REPAIR LOCATE RID REPLACE	DA/UTUT	
REPAIR LOCATE TABLESPACE DUMP or VERIFY	DW/UTRO	
REPAIR LOCATE TABLESPACE REPLACE	DA/UTUT	
REPAIR LOCATE INDEX PAGE DUMP or VERIFY		DW/UTRO
REPAIR LOCATE INDEX PAGE DELETE		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
- Blank - Object is not affected by this utility.

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

Table 66 (Page 1 of 2). Compatibility with REPAIR, LOCATE by KEY or RID

	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

Table 66 (Page 2 of 2). Compatibility with REPAIR, LOCATE by KEY or RID

	DUMP or VERIFY	DELETE or REPLACE
RECOVER INDEX	No	No
Note: RECOVER INDEX is compatible with LOCATE by RID, DUMP or VERIFY.		
RECOVER TABLESPACE	No	No
REORG INDEX	No	No
Note: REORG INDEX is compatible with LOCATE by RID, DUMP, VERIFY, or REPLACE.		
REORG TABLESPACE UNLOAD CONTINUE or PAUSE	No	No
REORG TABLESPACE UNLOAD ONLY or EXTERNAL	Yes	No
REPAIR DUMP or VERIFY	Yes	No
REPAIR DELETE or REPLACE	No	No
Note: REPAIR LOCATE INDEX PAGE REPLACE is compatible with LOCATE by RID REPLACE.		
REPORT	Yes	Yes
RUNSTATS INDEX SHRLEVEL REFERENCE	Yes	No
RUNSTATS INDEX SHRLEVEL CHANGE	Yes	Yes
RUNSTATS TABLESPACE	Yes	No
STOSPACE	Yes	Yes

Table 67 (Page 1 of 2). Compatibility with REPAIR, LOCATE by PAGE

	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

Table 67 (Page 2 of 2). Compatibility with REPAIR, LOCATE by PAGE

	TABLESPACE DUMP or VERIFY	TABLESPACE REPLACE	INDEX DUMP or VERIFY	INDEX REPLACE
RECOVER TABLESPACE (no option)	No	No	Yes	Yes
RECOVER TABLESPACE TOCOPY or TORBA	No	No	No	No
RECOVER TABLESPACE ERROR RANGE	No	No	Yes	Yes
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 DUMP or VERIFY	Yes	No	Yes	No
REPAIR DELETE or REPLACE	No	No	No	No
Note: REPAIR LOCATE INDEX PAGE REPLACE is compatible with LOCATE TABLESPACE PAGE.				
REPORT	Yes	Yes	Yes	Yes
RUNSTATS INDEX	Yes	Yes	Yes	No
RUNSTATS TABLESPACE	Yes	No	Yes	Yes
STOSPACE	Yes	Yes	Yes	Yes

Reviewing REPAIR output

The potential output from the REPAIR utility consists of a modified page or 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 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 carry out some checking of data. For example, REPAIR is unable to write a page with the wrong page number, DB2 will abend 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 will issue 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 caused by running REPAIR. These violations cause the target table space to be placed in the CHECK pending status. See “Chapter 2-4. CHECK DATA” on page 55 for information about resetting this status.

Sample control statements

Example 1: Replacing damaged data and verifying replacement. Repair the specified page of table space DSN8S61E. Verify that, at the specified offset (50), the damaged data (A00) is found. Replace it with the desired data (D11). Initiate a dump beginning at offset 50, for 4 bytes, to verify the replacement.

```
//STEP1    EXEC  DSNUPROC,UID='IUIQU1UH',UTPROC='',SYSTEM='V61A'
//SYSIN DD *
REPAIR OBJECT
  LOCATE TABLESPACE DSN8D61A.DSN8S61D PAGE X'02'
    VERIFY OFFSET 50 DATA X'A00'
    REPLACE OFFSET 50 DATA X'D11'
    DUMP OFFSET 50 LENGTH 4
```

Example 2: Removing a nonindexed row found by REORG. When reorganizing table space DSNDB04.TS1, you received the following message:

```
DSNU3401 DSNURBXA - ERROR LOADING INDEX, DUPLICATE KEY
                    INDEX = EMPINDEX
                    TABLE = EMP
                    RID OF INDEXED ROW  = X'00000201'
                    RID OF NONINDEXED ROW = X'00000503'
```

To resolve this error message, delete the nonindexed row and log the change. (The LOG keyword is not required; it is logged by default.)

```
REPAIR
  LOCATE TABLESPACE DSNDB04.TS1 RID (X'00000503')
    DELETE
```

Example 3: Report whether catalog and directory DBDs differ. Determine if the DBDs in the DB2 catalog and the DB2 directory are consistent for database DSN8D2AP.

```
REPAIR DBD TEST DATABASE DSN8D2AP
```

Example 4: Report differences between catalog and directory DBDs. After running the TEST option on database DSN8D2AP, and determining that the DBDs are inconsistent, determine the differences between the DBDs.

```
REPAIR DBD DIAGNOSE DATABASE DSN8D2AP OUTDDN SYSREC
```

Example 5: REPAIR table space with orphan row. After running DSN1CHKR on table space SYSDBASE, you received the following message:

```
DSN1812I ORPHAN ID = 20 ID ENTRY = 0190 FOUND IN
         PAGE = 000024
```

From a DSN1PRNT of page X'000024' and X'00002541', you identify that RID X'00002420' has a forward pointer of X'00002521'.

Repair the table space as follows:

1. Set the orphan's backward pointer to zeros.

```
REPAIR OBJECT LOG YES
 LOCATE TABLESPACE DSNDB06.SYSDBASE RID X'00002420
  VERIFY OFFSET X'0A' DATA X'00002422'
  REPLACE OFFSET X'0A' DATA X'00000000'
```

   Setting the pointer to zeros prevents the next step from updating link pointers while deleting, which can cause DB2 to abend if the orphan's pointers are incorrect.

2. Delete the orphan.

```
REPAIR OBJECT LOG YES
 LOCATE TABLESPACE DSNDB06.SYSDBASE RID X'00002420'
  VERIFY OFFSET X'06' DATA X'00002521'
  DELETE
```

Chapter 2-18. REPORT

The REPORT online 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. Use REPORT RECOVERY to find information necessary for recovering a table space, index, or a table space and all of its indexes. The REPORT utility also provides the LOB table spaces associated with a base table space.

For a diagram of REPORT syntax and a description of available options, see “Syntax and options of the control statement” on page 366. For detailed guidance on running this utility, see “Instructions for running REPORT” on page 368.

Output: The output from REPORT TABLESPACESET consists of the names of all table spaces in the table space set you specify. It also lists all tables in the table spaces and all tables 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 affects the recoverability of an index. For more information about this situation, see “Setting and clearing the informational COPY pending status” on page 103.

In a data sharing environment, the output provides:

- The RBA when migrated to Version 6
- The high and low RBA values of the migrated member
- A list of any SYSLGRNX records from 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 that the image copy was deleted from

Authorization required: To execute this utility, the privilege set of the process must include one of the following:

- 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:

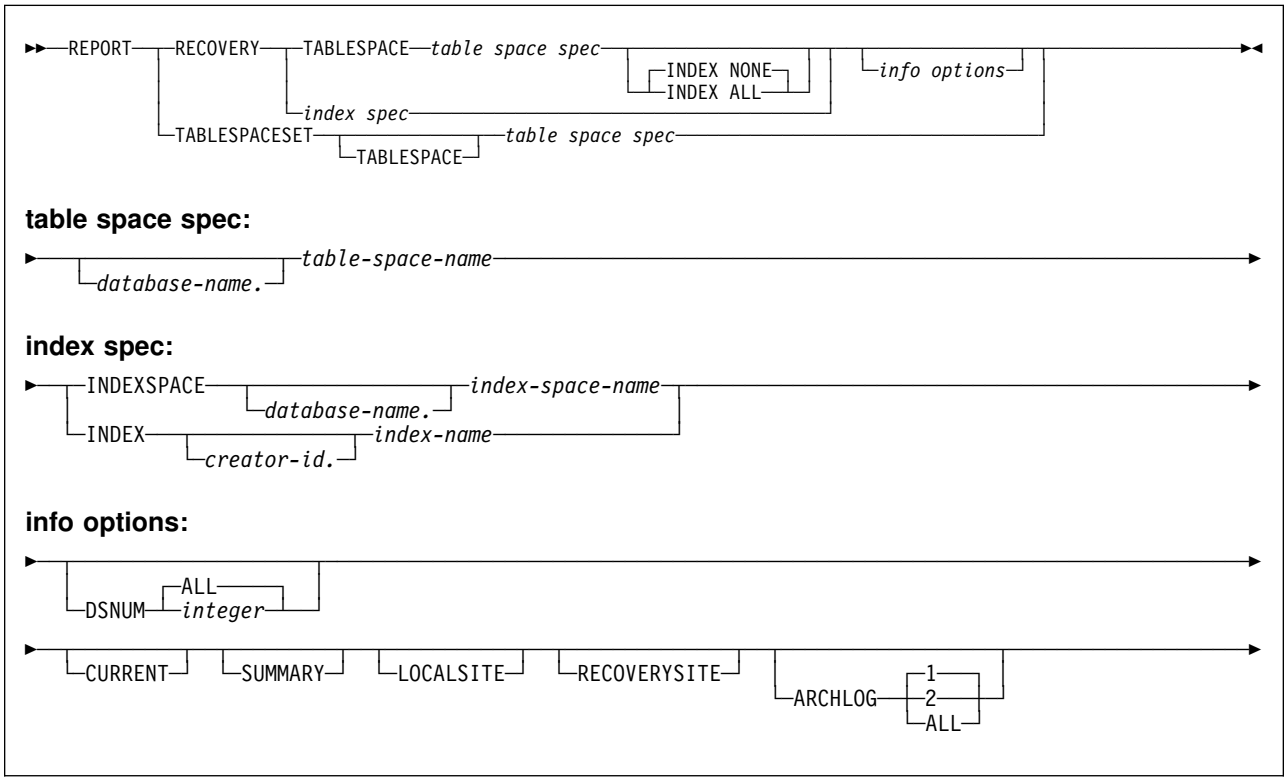
Phase	Description
UTILINIT	Initialization and setup
REPORT	Information collection
UTILTERM	Cleanup

Syntax and options of the control statement

The utility control statement defines the function 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

For guidance in interpreting syntax diagrams, see "How to read the syntax diagrams" on page 4.



Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see "Control statement coding rules" on page 27.

RECOVERY Indicates that recovery information for the specified table space or index is to be reported.

TABLESPACESET
Indicates that the names of all table spaces in the table space set, as well as the names of all indexes over tables in the table space set are to be reported.

TABLESPACE *database-name.table-space-name*
For REPORT RECOVERY, specifies the table space (and, optionally, the database to which it belongs) 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 the table space belongs to.

table-space-name Specifies the table space.

INDEXSPACE *database-name.index-space-name*
Specifies the index space being reported.

database-name Optionally specifies the database the index space belongs to.

index-space-name Specifies the index space name for the index being reported.

INDEX *creator-id.index-name*
Specifies the index in the index space being reported.

creator-id Optionally specifies the creator of the index.

index-name Specifies the index name to be reported.

The following keywords are optional:

INDEX NONE Does not report recovery information for index spaces associated with the specified table space.

INDEX ALL Reports recovery information for index spaces associated with the specified table space.

DSNUM For a table space, identifies a partition or data set, for which information is to be reported; or it reports information for the entire table space. For an index space, identifies a partition for which information is to be reported; or it reports information for the entire index space.

ALL Reports information 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 254.

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 as cataloged in the VSAM catalog. The data set name has this format:

catname.DSNDB*x*.*dbname*.*tsname*.I0001.A*nnn*

where:

catname The VSAM catalog name or alias
x C or D
dbname The database name
tsname The table space name
nnn The data set integer.

CURRENT Specifies that only the entries since the last recoverable point of the table space are to be reported. The last recoverable point is the last full image copy, LOAD REPLACE LOG YES or REORG LOG YES. If you specify DSNUM ALL, then the last recoverable point is a full image copy that was taken for the entire table space or index space.

If you do not specify CURRENT or if no last recoverable point exists, all SYSCOPY and SYSLGRNX entries for that table space or index space are reported, including those on archive logs. However, if you specify the CURRENT option, but the last recoverable point does not exist on the active log, DB2 prompts you to mount archive tapes until this point is found.

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 a report of all SYSCOPY records copied from a local site system.

RECOVERYSITE
Specifies a report of all SYSCOPY records copied for the recovery site system.

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 Optionally reports both archive log data sets 1 and 2.

Instructions for running REPORT

To run REPORT, you must:

1. Prepare the necessary data sets, as described in “Data sets used by REPORT” on page 369.
2. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27. (For examples of JCL for REPORT, see “Sample control statements” on page 374.)
3. Prepare a utility control statement, specifying the options for the tasks you want to perform, as described in “Instructions for specific tasks” on page 369. (For a complete description of the syntax and options for REPORT, see “Syntax and options of the control statement” on page 366.)
4. Check the compatibility table in “Concurrency and compatibility” on page 371 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 371.
6. Run REPORT.

See “Chapter 2-1. Invoking DB2 online utilities” on page 27 for an explanation of ways to execute DB2 utilities.

Data sets used by REPORT

Table 68 describes the data sets used by REPORT. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 68. Data sets used by REPORT

Data Set	Description	Required?
SYSIN	Input data set containing 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 card in the JCL:

Table space
Object to be reported. It is named in the REPORT control statement and is accessed through the DB2 catalog.

Creating the control statement

See “Syntax and options of the control statement” on page 366 for REPORT syntax and option descriptions. See “Sample control statements” on page 374 for examples of REPORT usage.

Instructions for specific tasks

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

Reporting recovery information

You can use the REPORT utility in planning for recovery. REPORT provides information necessary for recovering a table space. You can request report information for LOCALSITE or 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. It also indicates 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 denotes any non-COPY entries it finds in the SYSCOPY catalog table with asterisks. For example, an entry added by the QUIESCE utility is marked with asterisks in the REPORT output.

The following statement reports the names of all table spaces in the table space set containing table space DSN8S61E:

```
REPORT TABLESPACESET TABLESPACE DSN8D61A.DSN8S61E
```

The following statement reports recovery information for table space DSN8S61D for the local subsystem only:

```
REPORT RECOVERY TABLESPACE DSN8D61A.DSN8S61D LOCALSITE
```

For image copies of partitioned table spaces taken with the DSNUM ALL option, we recommend that you run REPORT RECOVERY DSNUM ALL. If you run REPORT RECOVERY DSNUM ALL CURRENT, DB2 reports additional historical information dating back to the last full image copy taken for the entire table space.

The REPORT RECOVERY utility output indicates if any image copies are unusable; image copies taken prior to REORG or LOAD events to reset REORG pending status are marked as unusable. REPORT output indicates which image copies will reset the REORG pending status by displaying the ICTYPE field as <R>, as shown in Figure 21.

```
DSNU582I = DSNUPPCP - REPORT RECOVERY TABLESPACE LDB1.TS1 SYSCOPY ROWS
TIMESTAMP = 1998-09-10-08.45.38.522996,  IC TYPE = <R>, SHR LVL = ,  DSNUM    = 0000,  START LRSN =AAE19AEEEOD4
DEV TYPE  =         ,                    IC BACK =   ,  STYPE   = A, FILE SEQ = 0000,  PIT LRSN  = 000000000000
LOW DSNUM = 0000,                        HIGH DSNUM = 0000
DSNAME    = LDB1.TS1                                      ,   MEMBER NAME =
LOWDSNUM = 0001 HIGHDSNUM = 0005
```

Figure 21. Example of REPORT RECOVERY unusable copy indication

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.SYSUTIL, 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 recoverable point does not exist on the active log, DB2 prompts you to mount archive tapes until this point is found.

REPORT TABLESPACESET can be used on the DB2 catalog and directory table spaces.

Terminating or restarting REPORT

You can restart a REPORT utility job, but it starts from the beginning again.

You can terminate REPORT with the TERM UTILITY command.

For guidance in restarting online utilities, see “Restarting an online utility” on page 48.

Concurrency and compatibility

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 you specify. It also specifies all tables in the table spaces, and specifies all tables dependent on those tables, including LOB table spaces.

For example, REPORT TABLESPACESET TABLESPACE LDB1.TS1 generates the following output:

```
DSNU000I   DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = RECTS004.CFD1
DSNU050I   DSNUGUTC - REPORT TABLESPACESET TABLESPACE LDB1.TS1
DSNU587I - DSNUPSET - REPORT TABLESPACE SET WITH TABLESPACE LDB1.TS1
                  TABLESPACE            TABLE                          DEPENDENT TABLE
                  LDB1.TS1              SYSADM.T1
                                        SYSADM.T2
DSNU580I   DSNUPORT - REPORT UTILITY COMPLETE - ELAPSED TIME=00:00:00
DSNU010I   DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 22. Example of REPORT TABLESPACESET

REPORT RECOVERY output: REPORT RECOVERY displays all the information about the image copy data sets and archive log data set that might be required during the recover.

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 3 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 describing 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 there is no data to report for one or more of these topics, the corresponding sections of the report contain this message:

```
DSNU588I - NO DATA TO BE REPORTED
```

Figure 23 on page 372 is a sample of REPORT RECOVERY output in a data sharing environment.

```
DSNU000I   DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = RECTS004.CFD1
DSNU050I   DSNUGUTC - REPORT RECOVERY TABLESPACE LDB1.TS1
DSNU581I - DSNUPREC - REPORT RECOVERY TABLESPACE LDB1.TS1
DSNU593I - DSNUPREC -  REPORT RECOVERY ENVIRONMENT RECORD:
'             MINIMUM   RBA: 000000000000
'             MAXIMUM   RBA: 000001DD9AE8
'             MIGRATING RBA: 000001DD9AE8
DSNU582I - DSNUPPCP - REPORT RECOVERY TABLESPACE LDB1.TS1 SYSCOPY ROWS
TIMESTAMP = 1998-09-10-08.21.25.912161,  IC TYPE = *Y*,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =AAE19AEEE0D4
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
LOW DSNUM = 0000,                        HIGH DSNUM = 0000
DSNAME    = LDB1.TS1                                    ,   MEMBER NAME = V61A

TIMESTAMP = 1998-09-10-08.21.46.464782,  IC TYPE =  F ,  SHR LVL = R,  DSNUM    = 0000,  START LRSN =AAE19B0B107B
DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
LOW DSNUM = 0000,                        HIGH DSNUM = 0000
DSNAME    = CPYLPF1                                     ,   MEMBER NAME = V61A

TIMESTAMP = 1998-09-10-08.22.07.674391,  IC TYPE = *Q*,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =AAE19B1EBCAF
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = W,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
LOW DSNUM = 0000,                        HIGH DSNUM = 0000
DSNAME    = LDB1.TS1                                    ,   MEMBER NAME = V61A

TIMESTAMP = 1998-09-10-08.22.15.786373,  IC TYPE =  I ,  SHR LVL = R,  DSNUM    = 0000,  START LRSN =AAE19B24CC8A
DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
LOW DSNUM = 0000,                        HIGH DSNUM = 0000
DSNAME    = CPYLPF2                                     ,   MEMBER NAME = V61A

TIMESTAMP = 1998-09-10-08.22.22.810734,  IC TYPE = *Q*,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =AAE19B3C986B
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = W,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
LOW DSNUM = 0000,                        HIGH DSNUM = 0000
DSNAME    = LDB1.TS1                                    ,   MEMBER NAME = V61A

TIMESTAMP = 1998-09-10-08.22.35.623910,  IC TYPE =  I ,  SHR LVL = R,  DSNUM    = 0000,  START LRSN =AAE19B4342EC
DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
LOW DSNUM = 0000,                        HIGH DSNUM = 0000
DSNAME    = CPYLPF3                                     ,   MEMBER NAME = V61A
DSNU586I - DSNUPSUM - REPORT RECOVERY TABLESPACE LDB1.TS1 SUMMARY
DSNU588I - DSNUPSUM - NO DATA TO BE REPORTED

DSNU592I - DSNUPREC - REPORT RECOVERY INFORMATION FOR DATA SHARING MEMBER : V61A
DSNU583I - DSNUPPLR - SYSLGRNX ROWS FROM REPORT RECOVERY FOR TABLESPACE LDB1.TS1
 UCDATE     UCTIME      START RBA      STOP  RBA     START LRSN     STOP  LRSN    PARTITION   MEMBER ID
 091098    16375569   000002017A57   000002019033   AAE19AD84EB1   AAE19AD8BA67     0000         0001
 091098    16380534   00000201C61C   000002022F16   AAE19AE1822F   AAE19AEE6DE6     0000         0001
 091098    16382567   AAE19AF4653C   AAE19AF4653C   AAE19AF4653C   AAE19AF4653C     0000         0001
 091098    16390230   0000020297AB   00000202FA14   AAE19B17C9D3   AAE19B1E7E32     0000         0001
 091098    16392787   000002034448   0000020663CE   AAE19B3037E6   AAE19B37FE01     0000         0001


DSNU586I - DSNUPSUM - REPORT RECOVERY TABLESPACE LDB1.TS1 SUMMARY
DSNU588I - DSNUPSUM - NO DATA TO BE REPORTED

DSNU592I - DSNUPREC - REPORT RECOVERY INFORMATION FOR DATA SHARING MEMBER : V61B


DSNU586I - DSNUPSUM - REPORT RECOVERY TABLESPACE LDB1.TS1 SUMMARY
DSNU588I - DSNUPSUM - NO DATA TO BE REPORTED

DSNU592I - DSNUPREC - REPORT RECOVERY INFORMATION FOR DATA SHARING MEMBER : V61C
```

Figure 23 (Part 1 of 2). Example of REPORT RECOVERY

```
DSNU586I - DSNUPSUM - REPORT RECOVERY TABLESPACE LDB1.TS1 SUMMARY
DSNU588I - DSNUPSUM - NO DATA TO BE REPORTED
DSNU589I - DSNUPREC - REPORT RECOVERY TABLESPACE LDB1.TS1 COMPLETE
DSNU580I   DSNUPORT - REPORT UTILITY COMPLETE - ELAPSED TIME=00:00:00
DSNU010I   DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 23 (Part 2 of 2). Example of REPORT RECOVERY

Figure 24 is a sample of REPORT RECOVERY TABLESPACE ARCHLOG output.

```
DSNU000I   DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = REPORT
DSNU050I   DSNUGUTC - REPORT RECOVERY TABLESPACE MYDBASE.MYSPACE ARCHLOG ALL
DSNU581I - DSNUPREC - REPORT RECOVERY TABLESPACE MYDBASE.MYSPACE
DSNU593I - DSNUPREC -  REPORT RECOVERY ENVIRONMENT RECORD:
'             MINIMUM   RBA: 000000000000
'             MAXIMUM   RBA: FFFFFFFFFFFF
'             MIGRATING RBA: 000000000000
DSNU582I - DSNUPPCP - REPORT RECOVERY TABLESPACE MYDBASE.MYSPACE SYSCOPY ROWS
TIMESTAMP = 1998-12-11-09.38.21.734394,  IC TYPE = *Y*,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =00000264B9A
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
LOW DSNUM = 0000,                        HIGH DSNUM = 0000
DSNAME    = MYCOPYA                                    ,   MEMBER NAME =
DS VOLSER = SCR03 ,

TIMESTAMP = 1998-12-11-09.38.48.913881,  IC TYPE =  F ,  SHR LVL = R,  DSNUM    = 0000,  START LRSN =00000266354
DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
LOW DSNUM = 0000,                        HIGH DSNUM = 0000
DSNAME    = MYCOPYIA                                   ,   MEMBER NAME = V61A
DS VOLSER = SCR03 ,

TIMESTAMP = 1998-12-11-09.39.09.542154,  IC TYPE = *Q*,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =000002674A2
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = W,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
LOW DSNUM = 0000,                        HIGH DSNUM = 0000
DSNAME    = MYCOPYIB                                   ,   MEMBER NAME = V61A
DS VOLSER = SCR03 ,

DSNU586I - DSNUPSUM - REPORT RECOVERY TABLESPACE MYDBASE.MYSPACE SUMMARY
              IC COPY VOLSER(S)            SCR03

DSNU583I - DSNUPPLR - SYSLGRNX ROWS FROM REPORT RECOVERY FOR TABLESPACE MYDBASE.MYSPACE

      START RBA      STOP  RBA     START LRSN     STOP  LRSN    PARTITION   MEMBER ID
    00000262E3A6   00000262E86F   00000262E3A6   00000262E86F     0000         0000
    000002632CC8   0000026394D4   000002632CC8   0000026394D4     0000         0000
    000002652EEF   0000026535A1   000002652EEF   0000026535A1     0000         0000
    000002671E1C   0000026724B9   000002671E1C   0000026724B9     0000         0000
    0000026792AA   000002679B08   0000026792AA   000002679B08     0000         0000

DSNU5834I - DSNUPPBS - REPORT RECOVERY TABLESPACE MYDBASE.MYSPACE ARCHLOG1 BSDS VO

      START RBA      END RBA       UNIT  VOLSER   DATA SET NAME
    000002328000   00000263DFFF   SYSDA  SCR03  DSNC420.ARCHLOG1.A0000003
    00000263E000   000002643FFF   SYSDA  SCR03  DSNC420.ARCHLOG1.A0000004
    000002644000   000002648FFF   SYSDA  SCR03  DSNC420.ARCHLOG1.A0000005  *
    000002649000   00000264CFFF   SYSDA  SCR03  DSNC420.ARCHLOG1.A0000006  *
    00000264D000   000002653FFF   SYSDA  SCR03  DSNC420.ARCHLOG1.A0000007  *
    000002654000   000002658FFF   SYSDA  SCR03  DSNC420.ARCHLOG1.A0000008  *
    000002659000   00000265EFFF   SYSDA  SCR03  DSNC420.ARCHLOG1.A0000009  *
    00000265F000   000002666FFF   SYSDA  SCR03  DSNC420.ARCHLOG1.A0000010  *
    000002667000   00000266BFFF   SYSDA  SCR04  DSNC420.ARCHLOG1.A0000011  *
    00000266C000   00000267BFFF   SYSDA  SCR04  DSNC420.ARCHLOG1.A0000012  *
```

Figure 24 (Part 1 of 2). Example of REPORT RECOVERY TABLESPACE ARCHLOG

```
DSNU5834I - DSNUPPBS - REPORT RECOVERY TABLESPACE MYDBASE.MYSPACE ARCHLOG2 BSDS VO

      START RBA      END RBA       UNIT  VOLSER   DATA SET NAME
    000002328000   00000263DFFF   SYSDA  SCR03  DSNC420.ARCHLOG2.A0000003
    00000263E000   000002643FFF   SYSDA  SCR03  DSNC420.ARCHLOG2.A0000004
    000002644000   000002648FFF   SYSDA  SCR03  DSNC420.ARCHLOG2.A0000005  *
    000002649000   00000264CFFF   SYSDA  SCR03  DSNC420.ARCHLOG2.A0000006  *
    00000264D000   000002653FFF   SYSDA  SCR03  DSNC420.ARCHLOG2.A0000007  *
    000002654000   000002658FFF   SYSDA  SCR03  DSNC420.ARCHLOG2.A0000008  *
    000002659000   00000265EFFF   SYSDA  SCR03  DSNC420.ARCHLOG2.A0000009  *
    00000265F000   000002666FFF   SYSDA  SCR03  DSNC420.ARCHLOG2.A0000010  *
    000002667000   00000266BFFF   SYSDA  SCR04  DSNC420.ARCHLOG2.A0000011  *
    00000266C000   00000267BFFF   SYSDA  SCR04  DSNC420.ARCHLOG2.A0000012  *

SNU586I - DSNUPSUM - REPORT RECOVERY TABLESPACE MYDBASE.MYSPACE SUMMARY
              ARCHLOG1 BSDS VOLSERS(S)      SCR03 *
                                            SCR04 *
              ARCHLOG2 BSDS VOLSERS(S)      SCR03 *
                                            SCR04 *

DSNU589I - DSNUPREC - REPORT RECOVERY TABLESPACE MYDBASE.MYSPACE COMPLETE
DSNU580I   DSNUPORT - REPORT UTILITY COMPLETE - ELAPSED TIME=00:00:00
DSNU010I   DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Figure 24 (Part 2 of 2). Example of REPORT RECOVERY TABLESPACE ARCHLOG

Sample control statements

Example 1: Sample JCL for REPORT RECOVERY.

```
//STEP1    EXEC DSNUPROC,UID='IUKUU206.REPORT2',
//            UTPROC='',SYSTEM='V61A'
//SYSIN    DD *
REPORT RECOVERY
       TABLESPACE DSN8D61A.DSN8S61E
//*
```

Example 2: Sample control statement for REPORT TABLESPACESET.

```
REPORT TABLESPACESET
       TABLESPACE UTQPD22A.UTQPS22E
```

Example 3: REPORT referentially related table spaces. The following statement reports the names of all table spaces in the table space set containing table space DSN8D61A.DSN8S61E.

```
REPORT TABLESPACESET TABLESPACE DSN8D61A.DSN8S61E
```

Example 4: REPORT RECOVERY information for a table space. This statement reports recovery information for table space DSN8D61A.DSN8S61D.

```
REPORT RECOVERY TABLESPACE DSN8D61A.DSN8S61D DSNUM ALL
```

Chapter 2-19. RUNSTATS

The RUNSTATS online utility gathers summary information about the characteristics of data in table spaces, indexes, and partitions. DB2 records this information in the DB2 catalog and uses it to select access paths to data during the bind process. It is available to the database administrator for evaluating database design and to aid in determining when table spaces or indexes must be reorganized.

There are two formats for the RUNSTATS utility: RUNSTATS TABLESPACE and RUNSTATS INDEX. RUNSTATS TABLESPACE gathers statistics on a table space and, optionally, on indexes or columns; RUNSTATS INDEX gathers statistics only on indexes.

| Use the STATISTICS keyword with LOAD, REBUILD INDEX, and REORG jobs to
| eliminate the need to execute RUNSTATS for updating catalog statistics. If you
| restart a LOAD or REBUILD INDEX job that uses the STATISTICS keyword, inline
| statistics collection does not occur. To update catalog statistics, run the
| RUNSTATS utility after the restarted utility job completes. For information about
| restarting a REORG job which uses the STATISTICS keyword, see page 328.

| You can specify that a LOB table space is to have space statistics collected so you
| can determine when the LOB table space should be reorganized. No statistics on
| the LOB table space affect access path selection.

DB2 invalidates statements in the dynamic statement cache when you run RUNSTATS against 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 pick up the latest access path changes.

For a diagram of RUNSTATS syntax and a description of available options, see “Syntax and options of the control statement” on page 376. For detailed guidance on running this utility, see “Instructions for running RUNSTATS” on page 382.

Output: RUNSTATS updates the DB2 catalog with table space or index space statistics or prints a report. The information updated by RUNSTATS is used by DB2 to select access paths to the data. You can query the catalog tables to obtain the updated statistics. See “Reviewing RUNSTATS output” on page 387 for a list of all the catalog tables and columns updated by RUNSTATS.

Additional information:

- 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 RUNSTATS, but only on a table space in the DSNDB06 database.

To use REPORT YES, you must have the SELECT privilege on the tables reported. Values are not reported from the tables the user is not authorized to see.

Execution phases of RUNSTATS: The RUNSTATS utility operates in these phases:

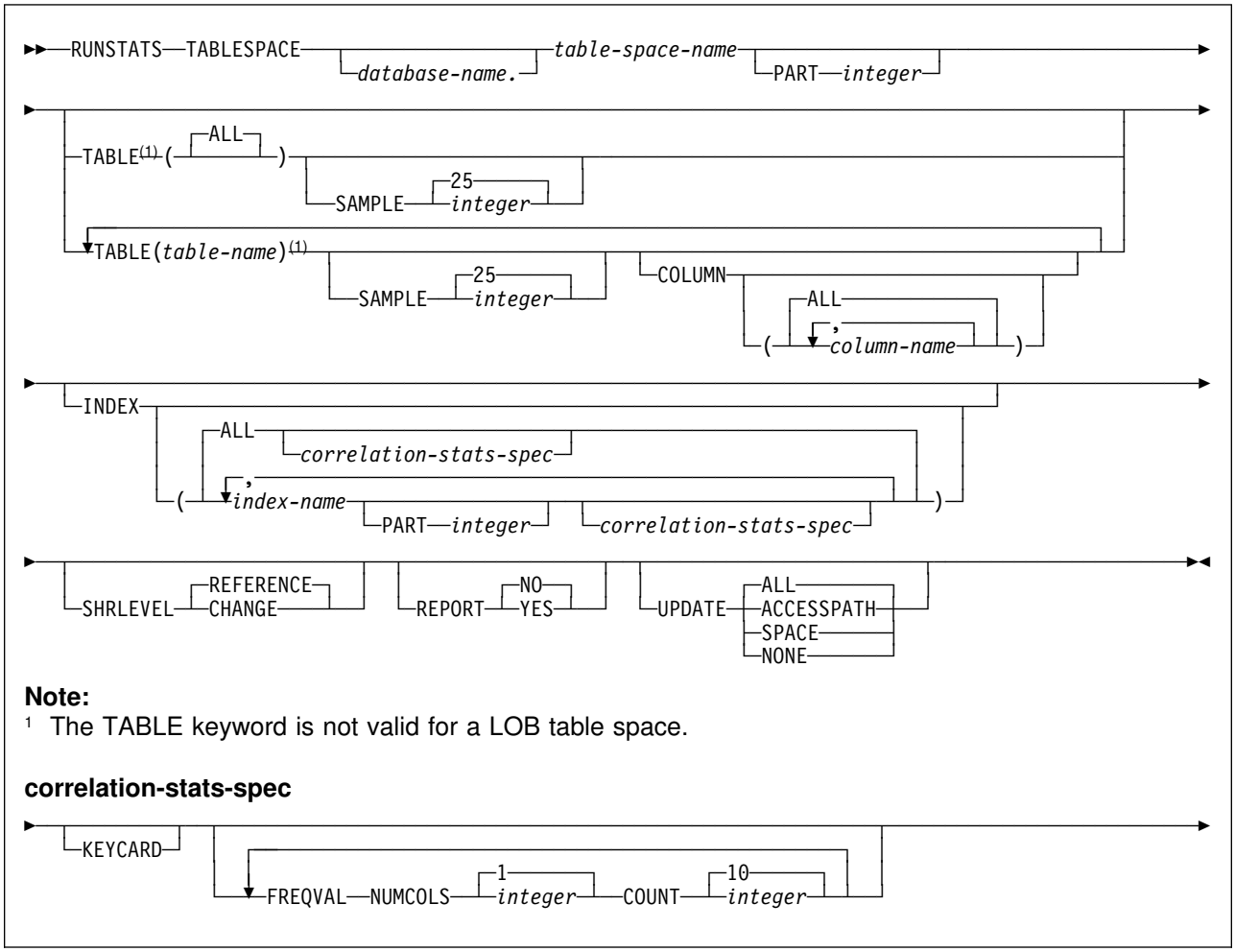
Phase	Description
UTILINIT	Initialization and setup
RUNSTATS	Scanning table space or index and updating catalog
UTILTERM	Cleanup

Syntax and options of the control statement

The utility control statement defines the function 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

For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.



Note:
1 The TABLE keyword is not valid for a LOB table space.

correlation-stats-spec

RUNSTATS TABLESPACE option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see "Control statement coding rules" on page 27.

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

Specifies the table space (and, optionally, the database to which it belongs) for which table space and table information is to be gathered. It must not be a table space in DSNDB01 or DSNDB07.

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 about which information is gathered.

If the table space specified by the TABLESPACE keyword is a LOB table space, these are the only keywords allowed: SHRLEVEL REFERENCE or CHANGE, REPORT YES or NO, and UPDATE ALL or NONE.

PART *integer*

Identifies a table space partition for 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 defined for the table space. The maximum is 254.

TABLE Specifies the table for which column information is to be gathered. All tables must belong to the table space 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**.

The TABLE option is not valid for a LOB table space.

SAMPLE *integer* Indicates the percentage of rows to sample when collecting non-indexed column statistics. Any value from 1 through 100 can be specified. The default is *25*.

The **SAMPLE** option is not allowed for LOB table spaces.

(*table-name***)** Specifies the tables for which column information is to be gathered. The parentheses are required. If you omit the qualifier, the user identifier for the utility job is used.

If you specify more than one table, you must repeat the TABLE option.

COLUMN Specifies columns for which column information is to be gathered.

You can only specify this option if you specify a 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 **COLUMN (ALL)** option is not allowed for LOB table spaces.

(*column-name***, ...)**
Specifies the columns for which statistics are to be gathered. The parentheses are required.

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, and might be gathered for additional index columns depending on the options you specify. All the indexes must be associated with the *same* table space, which must be the table space specified in the TABLESPACE option.

INDEX is valid for gathering statistics on an index on the auxiliary table.

(ALL) Specifies that the column information is to be gathered for all indexes defined on tables contained in the table space. The parentheses are required.

The **default** is **ALL**.

(*index-name***, ...)** Specifies the indexes for which information is to be gathered. The parentheses are required.

You can specify a list of index names. If you specify more than one index, separate each name with a comma.

PART *integer* Identifies an index partition for which statistics are to be collected.

integer is the number of the partition.

SHRLEVEL Tells 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, uncommitted data can be collected into statistical summaries.

REPORT Determines if a set of messages is generated to report the collected statistics.

NO Indicates that the set of messages is not output to SYSPRINT.

The **default** is **REPORT NO**.

YES Indicates that the set of messages is output to SYSPRINT. The messages generated are dependent on the combination of keywords (such as TABLESPACE, INDEX, TABLE, and COLUMN) 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 Tells whether the collected statistics are inserted into the catalog tables. UPDATE also allows you to select statistics used for access path selection or statistics used by database administrators.

ALL Indicates that all collected statistics will be updated in the catalog.

The **default** is **UPDATE ALL**.

ACCESSPATH Indicates that only the catalog table columns that provide statistics used for access path selection are updated.

SPACE Indicates that only the catalog table columns that provide statistics to help the database administrator assess the status of a particular table space or index are updated.

NONE Indicates that no catalog tables are updated with the collected statistics. This option is only valid when REPORT YES is specified.

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.

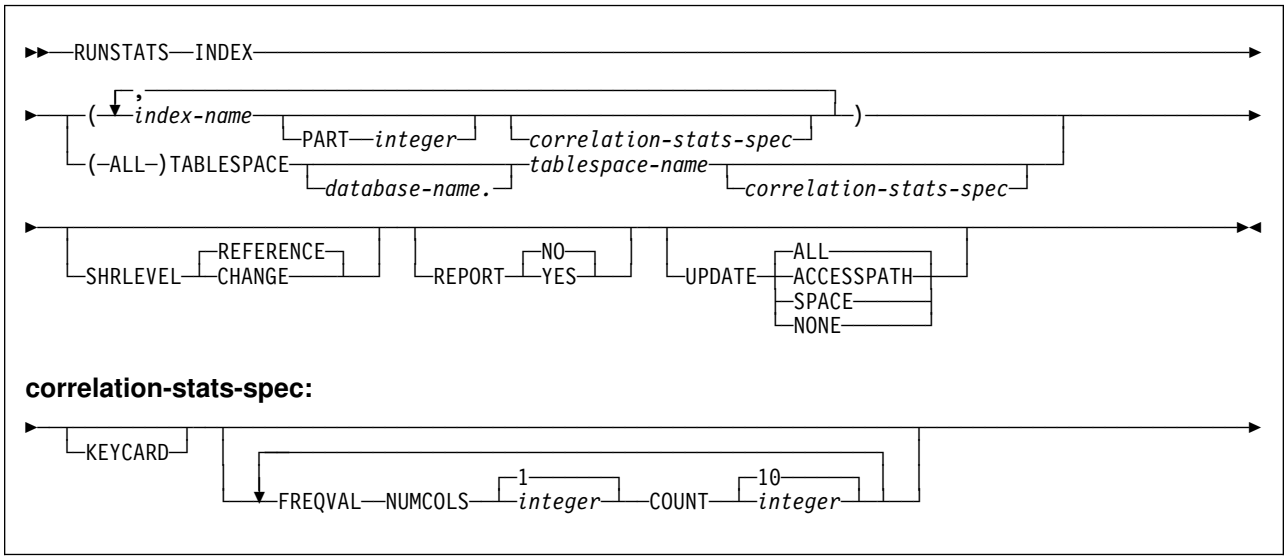
FREQVAL Controls the collection of frequent value statistics. If you specify FREQVAL, it must be followed by two additional keywords:

NUMCOLS Indicates the number of key columns to concatenate together when collecting frequent values from the specified index. Specifying '3' means to collect frequent values on the concatenation of the first three key

columns. The default is 1, which means collect frequent values on the first key column of the index.

COUNT Indicates the number of frequent values to be collected. Specifying '15' means collect 15 frequent values from the specified key columns. The default is *10*.

RUNSTATS INDEX syntax diagram



RUNSTATS INDEX option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see “Control statement coding rules” on page 27.

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.

(*index-name***, ...)** Specifies the indexes for which information is to be gathered. The parentheses are required.

You can specify a list of index names. If you specify more than one index, separate each name with a comma.

PART *integer* Identifies the index partition for which statistics are to be collected.

integer is the number of the partition.

(ALL) Specifies that information is to be gathered for all indexes defined on all tables in the specified table space.

TABLESPACE Names the table space and, optionally, the database it belongs to.

database-name The name of the database that the table space belongs to.

The **default** is **DSNDB04**.

tablespace-name The name of the table space.

SHRLEVEL Tells 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, uncommitted data can be used to collect statistical summaries.

REPORT Determines if a set of messages is generated to report the collected statistics.

NO Indicates that the set of messages is not output to SYSPRINT.

The **default** is **REPORT NO**.

YES Indicates that the set of messages is output to SYSPRINT. The messages generated are dependent on the combination of keywords (such as TABLESPACE, INDEX, TABLE, and COLUMN) 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 Tells whether the collected statistics are inserted into the catalog tables. UPDATE also allows you to select statistics used for access path selection or statistics used by database administrators.

ALL Indicates that all collected statistics will be updated in the catalog.

The **default** is **UPDATE ALL**.

ACCESSPATH Indicates that only the catalog table columns that provide statistics used for access path selection are updated.

SPACE Indicates that only the catalog table columns that provide statistics to help the database administrator assess the status of a particular table space or index are updated.

NONE Indicates that no catalog tables are updated with the collected statistics. This option is only valid when REPORT YES is specified.

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.

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 to concatenate together when collecting frequent values from the specified index. Specifying '3' means to collect frequent values on the concatenation of the first three key columns. The default is 1, which means collect frequent values on the first key column of the index.

COUNT Indicates the number of frequent values to be collected. Specifying '15' means collect 15 frequent values from the specified key columns. The default is *10*.

Instructions for running RUNSTATS

To run RUNSTATS, you must:

1. Read “Before running RUNSTATS” in this chapter.
2. Prepare the necessary data sets, as described in “Data sets used by RUNSTATS” on page 383.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27. (For examples of JCL for RUNSTATS, see “Sample control statements” on page 394.)
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 383. (For a complete description of the syntax and options for RUNSTATS, see “Syntax and options of the control statement” on page 376.)
5. Check the compatibility table in “Concurrency and compatibility” on page 385 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 385. RUNSTATS can be restarted, but it starts over again from the beginning.
7. Run RUNSTATS.

See “Chapter 2-1. Invoking DB2 online utilities” on page 27 for a description of ways to execute DB2 utilities.

Before running RUNSTATS

The columns RUNSTATS updates can be updated manually using SQL. Use caution when running RUNSTATS after another user has updated the statistical columns of the catalog. Because RUNSTATS puts information in these columns, values changed by the user are replaced.

Data sets used by RUNSTATS

Table 69 describes the data sets used by RUNSTATS. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 69. Data sets used by RUNSTATS

Data Set	Description	Required?
SYSIN	Input data set containing 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 cards in the JCL:

Table space or index
Object to be scanned. It is named in the RUNSTATS control statement and is accessed through the DB2 catalog.

Creating the control statement

See "Syntax and options of the control statement" on page 376 for RUNSTATS syntax and option descriptions. See "Sample control statements" on page 394 for examples of RUNSTATS usage.

Instructions for specific tasks

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

Deciding when to use RUNSTATS

DB2 uses the statistics generated by RUNSTATS to determine access paths to data. If no statistics are available, DB2 makes fixed default assumptions. To ensure the effectiveness of the paths selected, use RUNSTATS:

- After a table is loaded
- After an index is physically created
- After a table space is reorganized and inline statistics were not collected
- After there have been extensive updates, deletions, or insertions in a table space
- After you have run RECOVER TABLESPACE, REBUILD INDEX, or REORG INDEX, and you did not collect inline statistics with that utility
- Before running REORG with the OFFPOSLIMIT, INDREFLIMIT, or LEAFDISTLIMIT options.

Assessing table space status

Changes to a table space can also change its space requirements and performance. A database administrator can use RUNSTATS to assess the current status of the table space and help decide whether to reorganize or redesign the table space.

Updating statistics for a partitioned table space

If statistics do not exist for every partition, then RUNSTATS does not compute aggregate statistics (used for access path selection). After newly created partitioned table spaces have been loaded, run RUNSTATS on the entire table space (or on every partition) to take best advantage of access path selection.

Running RUNSTATS on the DB2 catalog

RUNSTATS may be used for the DB2 catalog, for index space and table space statistics. The following sample execution shows part of the output of RUNSTATS against a catalog table space and its indexes:

```
DSNU000I   DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = DSNTEX
DSNU050I   DSNUGUTC - RUNSTATS TABLESPACE DSNDB06.SYSBASE 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
```

DB2 uses the statistics collected on the catalog to determine the access path for user queries.

Improving performance

You can specify the STATISTICS keyword in LOAD, REBUILD INDEX, and REORG utility statements, which results in updated table space or index space catalog statistics for the objects the utility was run on. Another method of improving RUNSTATS performance is to specify the SAMPLE option on tablespaces that were defined with the LARGE option, which reduces the number of rows sampled for statistics.

When you run RUNSTATS concurrently against partitions of a partitioned table space or index, the sum of the processor time for the concurrent jobs will be roughly equivalent to the processor time it takes to run a single RUNSTATS job against the entire table space or index. However, the total elapsed time for the concurrent jobs can be significantly less than when you run RUNSTATS against an entire table space or index.

When requesting nonindexed column statistics, provide a list of columns that might be used in queries as search conditions in a WHERE clause. Collecting statistics on all columns of a table is costly and might not be necessary.

Terminating or restarting RUNSTATS

You can restart a RUNSTATS utility job, but it starts from the beginning again.

You can terminate RUNSTATS with the TERM UTILITY command.

For guidance in restarting online utilities, see “Restarting an online utility” on page 48.

Concurrency and compatibility

Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions of the same table space or index space are compatible.

Table 70 shows which claim classes RUNSTATS claims and drains and any restrictive state the utility sets on the target object.

Table 71 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 is also shown.

Table 70. Claim classes of RUNSTATS operations. Use of claims and drains; restrictive states set on the target object.

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]		
Index or partition			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
- Blank - 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 71 (Page 1 of 2). RUNSTATS compatibility

	RUNSTATS TABLESPACE SHRLEVEL REFERENCE	RUNSTATS TABLESPACE SHRLEVEL CHANGE	RUNSTATS INDEX SHRLEVEL REFERENCE	RUNSTATS INDEX SHRLEVEL CHANGE
CHECK DATA	No	No	No	No
CHECK INDEX	Yes	Yes	Yes	Yes
CHECK LOB	Yes	Yes	Yes	Yes

Table 71 (Page 2 of 2). RUNSTATS compatibility

	RUNSTATS TABLESPACE SHRLEVEL REFERENCE	RUNSTATS TABLESPACE SHRLEVEL CHANGE	RUNSTATS INDEX SHRLEVEL REFERENCE	RUNSTATS INDEX SHRLEVEL CHANGE
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	Yes	Yes	Yes	Yes
QUIESCE	Yes	Yes	Yes	Yes
REBUILD INDEX	Yes	Yes	No	No
RECOVER INDEX	Yes	Yes	No	No
RECOVER TABLESPACE (no options)	No	No	Yes	Yes
RECOVER TABLESPACE TOCOPY or TORBA	No	No	No	No
RECOVER ERROR RANGE	No	No	Yes	Yes
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 KEY or RID DELETE or REPLACE	No	No	No	Yes
REPAIR LOCATE TABLESPACE PAGE REPLACE	No	No	Yes	Yes
REPAIR LOCATE INDEX PAGE REPLACE	Yes	Yes	No	No
REPORT	Yes	Yes	Yes	Yes
RUNSTATS	Yes	Yes	Yes	Yes
STOSPACE	Yes	Yes	Yes	Yes

Reviewing RUNSTATS output

RUNSTATS alters the tables and columns in the DB2 catalog tables listed below. A report of statistics gathered during processing is generated with the REPORT YES option.

RUNSTATS sets the following columns to -1 for large table spaces.

- CARD in SYSTABLES
- CARD in SYSINDEXPART
- FAROFFPOS in SYSINDEXPART
- NEAROFFPOS in SYSINDEXPART
- FIRSTKEYCARD in SYSINDEXES
- FULLKEYCARD in SYSINDEXES

Index statistics and table space statistics: The following catalog tables are updated depending on the source of the statistics as well as the value of the UPDATE option.

Table 72 (Page 1 of 2). Catalog tables updated by RUNSTATS

Keyword	UPDATE Option	Catalog Table
TABLESPACE	UPDATE ALL	SYSTABLESPACE SYSTABLEPART[2] SYSTABLES[2] SYSTABSTATS[1,2] SYSLOBSTATS[3]
	UPDATE ACCESSPATH[2]	SYSTABLESPACE SYSTABLES
	UPDATE SPACE[2]	SYSTABSTATS[1] SYSTABLEPART SYSLOBSTATS
TABLE	UPDATE ALL	SYSCOLUMNS SYSCOLSTATS[1]
	UPDATE ACCESSPATH	SYSCOLUMNS SYSCOLSTATS[1]

Table 72 (Page 2 of 2). Catalog tables updated by RUNSTATS

Keyword	UPDATE Option	Catalog Table
INDEX	UPDATE ALL	SYSCOLUMNS SYSCOLDIST SYSCOLDISTSTATS[1] SYSCOLSTATS[1] SYSINDEXES SYSINDEXPART SYSINDEXSTATS[1]
	UPDATE ACCESSPATH	SYSCOLUMNS SYSCOLDIST SYSCOLDISTSTATS[1] SYSCOLSTATS SYSINDEXES SYSINDEXSTATS[1]
	UPDATE SPACE	SYSINDEXPART

Notes:

1. Only updated for partitioned cases. When you run RUNSTATS against single partitions of an object, the partition-level statistics that result are used to update the aggregate statistics for the entire object. The catalog tables containing these partition-level statistics are the following:
   - SYSCOLSTATS
   - SYSCOLDISTSTATS
   - SYSTABSTATS
   - SYSINDEXSTATS
2. If the specified table space is a LOB table space, this is not applicable.
3. Only applicable when the specified table space is a LOB table space.

Access path statistics

The catalog table columns listed in Table 73 are used by DB2 to select access paths to data during the bind process. Refer to Section 5 (Volume 2) of *DB2 Administration Guide* for further information regarding these columns.

Table 73 does not describe information about LOB columns, because those statistics are not used for access path selection. For indexes on auxiliary tables, only the NLEVELS and FIRSTKEYCARDF columns in SYSIBM.SYSINDEXES have an effect on the access path. For information on what values in these columns indicate for LOBs, see Appendix D of *DB2 SQL Reference*.

A value in the column “Use” 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 Appendix E, “Notices” on page 545.

Table 73 (Page 1 of 3). DB2 catalog columns used to select data access paths

Column Name	Column Description	Use
SYSTABLES		
CARDF	Total number of rows in the table.	S
NPAGES	Total number of pages on which rows of this table appear.	S

Table 73 (Page 2 of 3). DB2 catalog columns used to select data access paths

Column Name	Column Description	Use
PCTROWCOMP	Percentage of rows compressed within the total number of active rows in the table.	S
STATSTIME	The date and time when RUNSTATS was last executed to update the statistics.	G
SYSTABSTATS		
# CARD or CARDF	Total number of rows in the partition.	S
NPAGES	Total number of pages on which rows of this partition appear.	S
SYSCOLUMNS		
COLCARDF	Estimated number of distinct values for the column. For an indicator column, this is the number of non-zero length, non-null LOBs. The value is -1 if statistics have not been gathered. The value is -2 for columns of an auxiliary table.	S
HIGH2KEY	Second highest value of the column. Blank if statistics have not been gathered or 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 is an updatable column.	S
LOW2KEY	Second lowest value of the column. Blank if statistics have not been gathered or 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 is an updatable column.	S
STATSTIME	The date and time when RUNSTATS was last executed to update the statistics.	G
SYSCOLDIST		
CARDF	The number of distinct values for the column group. This number is only valid for cardinality (TYPE **C**) key column statistics.	S
COLGROUPCOLNO	Identifies the set of columns 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 specified in COLVALUE.	S
NUMCOLUMNS	The number of columns associated with the key column statistics	G
STATSTIME	The date and time when RUNSTATS was last executed to update the statistics.	G
SYSTABLESPACE		
# NACTIVE or # NACTIVEF	Number of active pages in the table space; shows the number of pages that are touched if a record cursor is used to scan the entire file. The value is -1 if statistics have not been gathered.	S
STATSTIME	The date and time when RUNSTATS was last executed to update the statistics.	G
DSSIZE	Maximum size of a data set in kilobytes.	G
SYSINDEXES		

Table 73 (Page 3 of 3). DB2 catalog columns used to select data access paths

Column Name	Column Description	Use
CLUSTERRATIOF	A number between 0 and 1 that when multiplied by 100 gives the percentage of rows 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% rows are in clustering order.	S
CLUSTERING	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
STATSTIME	The date and time when RUNSTATS was last executed to update the statistics.	G
SYSINDEXSTATS		
CLUSTERRATIOF	A number between 0 and 1 that when multiplied by 100 gives the percentage of rows 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% rows are in clustering order.	S
FIRSTKEYCARD or FIRSTKEYCARDF	For the index partition, number of distinct values of the first key column.	S
FULLKEYCARD or FULLKEYCARDF	For the index partition, number of distinct values of the key.	S
NLEAF	Number of active leaf pages in the index partition.	S
NLEVELS	Number of levels in the index tree in the partition.	S
KEYCOUNT or KEYCOUNTF	Total number of rows in the partition.	S

Space statistics (columns for tuning information)

The following catalog table columns are updated by RUNSTATS to help database administrators assess the status of a particular table space or index.

A value in the column “Use” 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 Appendix E, “Notices” on page 545.

Table 74 (Page 1 of 5). DB2 catalog columns for tuning information

Column Name	Column Description	Use
SYSTABLEPART		
CARD	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. The database administrator can validate design assumptions against this actual count. Over a period of time, it can show the rate of change or growth of the table space.	G
NEARINDREF	Number of rows relocated near their original page. See the description following FARINDREF.	S

Table 74 (Page 2 of 5). DB2 catalog columns for tuning information

Column Name	Column Description	Use
FARINDREF	Number of rows relocated far from their original page. If an update operation increases the length of a record by more than the amount of space available 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 less; 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 relocated near its original page tends to be accessed more quickly than one relocated far from its original page.	S
PAGESAVE	Percentage of pages 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 percent, so that the pages required are only 75 percent of what would be required without data compression. The value is 0 if there are no savings from using data compression, 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 bytes required for the dictionary, and the bytes required 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 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. A database administrator can use this figure to validate design assumptions, and tell how much of the space allocated to the table space is utilized. 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

Table 74 (Page 3 of 5). DB2 catalog columns for tuning information

Column Name	Column Description	Use
PERCDROP	For nonsegmented table spaces, the percentage of space occupied by rows of data from dropped tables. For segmented table spaces, this value is zero. After reorganization, this value is always zero. Space occupied by dropped tables is reclaimed by reorganization. Hence, this figure is one indicator of when a table space should be reorganized.	S
CARDF	Total number of rows in the table space or partition, or if the table space is a LOB table space, the number of LOBs in the table space. The value is -1 if statistics have not been gathered. The database administrator can validate design assumptions against this actual count. Over a period of time, it can show the rate of change or growth of the table space.	G
SPACE	The number of kilobytes of space currently allocated for all extents. A value of -1 indicates that the data set was defined with the DEFINE NO attribute, and the first insert operation has not occurred.	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
SYSINDEXPART		
CARDF	Number of rows referred to by the index or partition. Those figures, for all the partitions, tell the database administrator how the key ranges specified for each partition have divided the rows among the several partitions.	S

Table 74 (Page 4 of 5). DB2 catalog columns for tuning information

Column Name	Column Description	Use
NEAROFFPOSF	Number of times it would be necessary to access a different, “near-off” page when accessing all the data records in index order. Each time, it is probable that accessing the “next” record would require I/O activity. See the description following FAROFFPOS. NEAROFFPOS is incremented if the current indexed row is not on the same or next data page of the previous indexed row, and the distance between the two data pages does not qualify for FAROFFPOS. For nonsegmented table spaces, a page is considered near-off 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 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 contained in the segmented table space.Not applicable for the index on an auxiliary table (-1).	S
FAROFFPOSF	Number of times is would be necessary to access a different, “far-off” page when accessing all the data records in index order. Each time, it is almost certain that accessing the “next” record would require I/O activity. For nonsegmented table spaces, a page is considered far-off the present page if the two page numbers differ by 16 or more. For segmented table spaces, a page is considered far-off the present page if the two page numbers differ by SEGSIZE * 2 or more. Together, NEAROFFPOS and FAROFFPOS tell 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 was specified on the CREATE TABLESPACE statement, the NEAROFFPOS after reorganization reflects the table on which the index is defined. Optimal values should not be expected for nonclustering indexes. Not applicable for the index on an auxiliary table (-1).	S

Table 74 (Page 5 of 5). DB2 catalog columns for tuning information

Column Name	Column Description	Use
LEAFDIST	100 times the average distance in page IDs between successive leaf pages during a sequential access of the index. This value helps to tell how well an index is organized. The value is at its lowest just after the index has been reorganized. Changes increase it; and you can reduce it again by reorganizing the index, either explicitly or as part of a general table space reorganization.	S
SPACE	The number of kilobytes of space currently allocated for all extents. A value of -1 indicates that the data set was defined with the DEFINE NO attribute, and the first insert operation has not occurred.	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
SYSLOBSTATS		
FREESPACE	The number of kilobytes of available space in the LOB table space.	S
ORGRATIO	The ratio of organization in the LOB table space. A value of 1 indicates perfect organization of the LOB table space. The greater the value exceeds 1, the more disorganized the LOB table space.	S

After running RUNSTATS

After running RUNSTATS, rebind any application plans that use the tables or indexes so that they use the new statistics.

Sample control statements

Example 1: Update catalog statistics while allowing changes. Update the catalog statistic columns for table space DSN8S61E and all its associated indexes, sampling 25 percent of the rows. Permit other processes to make changes while this utility is executing.

```
//STEP1    EXEC DSNUPROC,UID='IUJQU225.RUNSTA',TIME=1440,
//         UTPROC='',
//         SYSTEM='V61A',DB2LEV=DB2A
//UTPRINT  DD  SYSOUT=*
//SYSIN    DD *
RUNSTATS TABLESPACE DSN8D61A.DSN8S61E
   TABLE(ALL) SAMPLE 25
    INDEX(ALL)
   SHRLEVEL CHANGE
```

Example 2: Update catalog statistics, do not allow updates. Update the catalog statistics for indexes XEMPL1 and XEMPL2. Do not permit other processes to change the table space associated with XEMPL1 and XEMPL2 (table space DSN8S61E) while this utility is executing.

```
RUNSTATS INDEX (DSN8610.XEMPL1,DSN8610.XEMPL2)
```

Example 3: Update index statistics. Obtain statistics on the index XEMPL1.

```
RUNSTATS INDEX (DSN8610.XEMPL1)
```

Example 4: Update statistics for several tables. Update the catalog statistics for all columns in the TCONA and TOPTVAL tables in table space DSN8D61P.DSN8S61C. Update the column statistics for the LINENO and DSPLINE columns in the TDSPTXT table in table space DSN8D61P.DSN8S61C.

```
RUNSTATS TABLESPACE(DSN8D61P.DSN8S61C) TABLE (TCONA)
                                       TABLE (TOPTVAL) COLUMN(ALL)
                                       TABLE (TDSPTXT) COLUMN(LINENO,DSPLINE)
```

Example 5: Update all statistics for a table space. Update all catalog statistics (table space, tables, columns, and indexes) for a table space.

```
RUNSTATS TABLESPACE(DSN8D61P.DSN8S61C) TABLE INDEX
```

Example 6: Update statistics used for access path selection. Update the catalog with *only* the statistics that are collected for access path selection. Report the collected statistics and route the statistics to SYSPRINT.

```
RUNSTATS TABLESPACE DSN8D61A.DSN8S61E
  REPORT YES
  UPDATE ACCESSPATH
```

Example 7: Update all statistics and generate report. Update the catalog with *all* the statistics (access path and space). Report the collected statistics and route the statistics to SYSPRINT.

```
RUNSTATS TABLESPACE DSN8D61A.DSN8S61E
  REPORT YES
  UPDATE ALL
```

Example 8: Report statistics without updating catalog. Do not update the catalog with the collected statistics. Report the collected statistics and route the statistics to SYSPRINT.

```
RUNSTATS TABLESPACE DSN8D61A.DSN8S61E
  REPORT YES
  UPDATE NONE
```

Example 9: Update statistics for a partition. Update the statistics for the table space and the partitioning index after a change to partition 1.

```
RUNSTATS TABLESPACE DSN8D61A.DSN8S61E PART 1 INDEX(DSN8610.XEMP1 PART 1)
```

Chapter 2-20. STOSPACE

The STOSPACE online utility updates DB2 catalog columns that indicate how much space is allocated for storage groups and related table spaces and indexes.

How to use this chapter: For a diagram of STOSPACE syntax and a description of available options, see "Syntax and options of the control statement." For detailed guidance on running this utility, see "Instructions for running STOSPACE" on page 398 .

Output: The output from STOSPACE consists of new values in a number of catalog tables. See "Reviewing STOSPACE output" on page 401 for a list of columns and tables STOSPACE updates.

Authorization required: To execute this utility, the privilege set of the process must include one of the following:

- STOSPACE privilege
- SYSCTRL or SYSADM authority

Execution phases of STOSPACE: The STOSPACE utility operates in these phases:

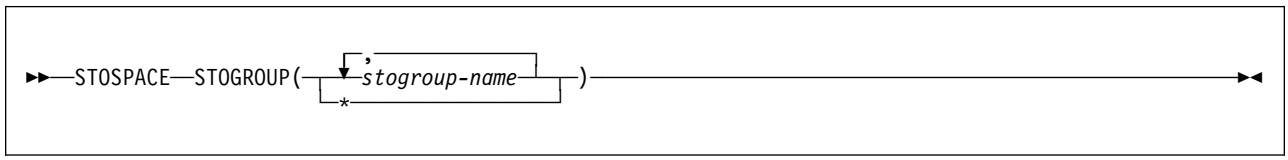
Phase	Description
UTILINIT	Initialization and setup
STOSPACE	
UTILTERM	Cleanup

Syntax and options of the control statement

The utility control statement defines the function 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

For guidance in interpreting syntax diagrams, see "How to read the syntax diagrams" on page 4.



Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see “Control statement coding rules” on page 27.

STOGROUP

Identifies the groups to be processed.

(*stogroup-name***, ...)** Is the name of a storage group. You can use a list of from one to eight storage group names. Separate items in the list by commas and enclose them in parentheses.

***** Processes all storage groups.

Instructions for running STOSPACE

To run STOSPACE, you must:

1. Prepare the necessary data sets, as described in “Data sets used by STOSPACE.”
2. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 online utilities” on page 27. (For examples of JCL for STOSPACE, see “Sample control statement” on page 402.)
3. Prepare a utility control statement, specifying the options for the tasks you want to perform, as described in “Instructions for specific tasks” on page 399. (For a complete description of the syntax and options for STOSPACE, see “Syntax and options of the control statement” on page 397.)
4. Check the compatibility rules in “Concurrency and compatibility” on page 401 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 401.
6. Run STOSPACE.

See “Chapter 2-1. Invoking DB2 online utilities” on page 27 for an explanation of ways to execute DB2 utilities.

Data sets used by STOSPACE

Table 75 describes the data sets used by STOSPACE. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 75. Data sets used by STOSPACE

Data Set	Description	Required?
SYSIN	Input data set containing 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 card in the JCL:

Storage group
Object to be reported. It is named in the STOSPACE control statement and is accessed through the DB2 catalog.

Creating the control statement

See “Syntax and options of the control statement” on page 397 for STOSPACE syntax and option descriptions. See “Sample control statement” on page 402 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 required by STOSPACE”
“Obtaining statistical information with STOSPACE”
“Understanding the values in a SPACE column” on page 400

Ensuring availability of objects required by STOSPACE

For each specified storage group, STOSPACE looks at the SYSIBM.SYSTABLESPACE and SYSIBM.SYSINDEXES catalog tables to tell which objects belong to that storage group. For each object, the amount of space allocated is determined from an appropriate VSAM catalog. Hence the table spaces and indexes need not 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 DBD for the objects involved. This requires that the appropriate database, table spaces, and index spaces not be in the stopped state.

Obtaining statistical information with STOSPACE

Table 76 lists statistical information recorded by the STOSPACE utility that is useful for space allocation decisions.

Table 76. DB2 catalog data collected by STOSPACE

Catalog Table	Column Name	Column Description
SYSTABLESPACE	SPACE	Number of kilobytes of storage allocated to the table space
SYSTABLEPART	SPACE	Number of kilobytes of storage allocated to the table space partition
SYSINDEXES	SPACE	Number of kilobytes of storage allocated to the index
SYSINDEXPART	SPACE	Number of kilobytes of storage allocated to the index partition
SYSSTOGROUP	SPACE	Number of kilobytes of storage allocated to the storage group
SYSSTOGROUP	SPCDATE	Date when STOSPACE was last run on a particular storage group
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 DASD space 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 SPACE column of the catalog represents the high allocated RBA of the VSAM linear data set. Use the value in the SPACE 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, it is a good idea to increase the primary quantity value for the data set and run the REORG utility.

For information about space utilization in the DSN8S61E table space in the DSN8D61A database, first run the STOSPACE utility, and then execute this SQL statement:

General-use Programming Interface

```
SELECT SPACE
  FROM SYSIBM.SYSTABLESPACE
  WHERE NAME = 'DSN8S61E
  AND DBNAME = 'DSN8D61A';
```

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 DSN8G610, as well as SYSIBM.SYSTABLESPACE and SYSIBM.SYSINDEXES for every table space and index belonging to DSN8G610, use the following utility:

```
STOSPACE STOGROUP DSN8G610
```

Understanding the values in a SPACE column

The value in a SPACE column is total allocated space, not only space allocated on the current list of volumes in the storage groups. Volumes can be deleted 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.

For each specified storage group, STOSPACE looks at the SYSIBM.SYSTABLESPACE and SYSIBM.SYSINDEXES catalog tables to tell which objects belong to that storage group. For each object, the amount of space allocated is determined from an appropriate VSAM catalog. Therefore, the table spaces and indexes need not be available to DB2 when STOSPACE is running; only the DB2 catalog and appropriate VSAM catalogs are needed. However, to gain access to the VSAM catalog, the utility must have available to it the DBD for the objects involved. This requires that the appropriate database, table spaces, and index spaces not be in the stopped state.

Considerations for running STOSPACE

For user-defined spaces, STOSPACE does not record any statistics.

Terminating or restarting STOSPACE

You can terminate STOSPACE with the TERM UTILITY command.

You can restart a STOSPACE utility job; however, it starts again from the beginning.

For more guidance in restarting online utilities, see “Restarting an online utility” on page 48.

Concurrency and compatibility

STOSPACE does not set a utility restrictive state on the target object.

STOSPACE can run concurrently on the same target object with any utility. However, because STOSPACE updates the catalog, concurrent STOSPACE utility jobs or other concurrent applications that update the catalog might cause timeouts and deadlocks.

Reviewing STOSPACE output

The output from STOSPACE consists of new values in the columns and tables listed. In each case, an amount of space is given in kilobytes.

- SPACE in SYSIBM.SYSINDEXES shows the amount of space allocated to indexes. If the index is not defined using STOGROUP, or STOSPACE has not been executed, the value is zero.
- SPACE in SYSIBM.SYSTABLESPACE shows the amount of space allocated to table spaces. If the table space is not defined using STOGROUP, or STOSPACE has not been executed, the value is zero.
- SPACE in SYSIBM.SYSINDEXPART shows the amount of space allocated to index partitions. If the partition is not defined using STOGROUP, or STOSPACE has not been executed, the value is zero.
- SPACE in SYSIBM.SYSTABLEPART shows the amount of space allocated to table partitions. If the partition is not defined using STOGROUP, or STOSPACE has not been executed, the value is zero.
- SPACE in SYSIBM.SYSSTOGROUP shows the amount of space allocated to storage groups.
- SPCDATE in SYSIBM.SYSSTOGROUP shows, in the form *yyddd*, the date when STOSPACE was last used on a particular storage group.
- STATSTIME in SYSIBM.SYSSTOGROUP shows the timestamp for the time.

Sample control statement

Example: Update catalog SPACE columns. Update the DB2 catalog SPACE columns for storage group DSN8G610.

```
//STEP1    EXEC DSNUPROC,UID='FUAUU330.STOSPCE',
//         UTPROC='',
//         SYSTEM='V61A'
//SYSIN    DD *
STOSPACE STOGROUP DSN8G610
//*
```

Section 3. Stand-alone utilities

Chapter 3-1. 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.

Creating utility statements and EXEC PARM parameters

Utility control statements and parameters define the function 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.

Creating utility control statements

You can 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

Control statement coding rules

Utility control statements are read from the DD name input stream. The statements in that stream must conform to these 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 being parsed. No concatenation character is necessary.
- The SYSIN stream can contain multiple utility control statements.

Specifying options using the JCL EXEC PARM parameter

The following stand-alone utilities obtain function options from the EXEC PARM parameter:

DSN1CHKR
DSN1COMP
DSN1COPY
DSN1PRNT

Following OS/390 JCL EXEC PARM specification rules

The parameters you specify must obey these OS/390 JCL EXEC PARM parameter specification rules:

- Enclose multiple subparameters in single quotes or parentheses, separating subparameters with commas. For example:

```
//name   EXEC PARM='ABC,...,XYZ'
```

- The total length cannot exceed 100 characters.
- Blanks are not allowed within the parameter specification.

To specify the parameter across multiple lines:

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.

For example:

```
//stepname  EXEC PARM=(ABC,...LMN,
           OPQ,...,XYZ)
```

Example of an option description

Where the syntax of each utility control statement is described, parameters are indented under the option keyword they must follow. Here is an example:

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)**.

In the example, AFTER is an option keyword, and *integer* is a parameter. Values of parameters are usually enclosed in parentheses. The syntax diagrams for utility control statements show parentheses where they are required.

Chapter 3-2. 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.

Before running DSNJLOGF

This section contains information you need to be aware of prior to running DSNJLOGF.

Environment

Run DSNJLOGF as an MVS job.

Control statement

See “Sample 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 to be preformatted. The data set must be an empty VSAM linear data set.

SYSPRINT Defines the print spool class or data set for print output. The logical record length (LRECL) is 132.

Sample control statement

The following sample control statement preformats the active log data sets.

```
//JOBLIB DD  DSN=DSN610.SDSNLOAD,DISP=SHR
//STEP1    EXEC PGM=DSNJLOGF
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSUT1   DD DSN=DSNC610.LOGCOPY1.DS01,DISP=SHR
//STEP2    EXEC PGM=DSNJLOGF
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSUT1   DD DSN=DSNC610.LOGCOPY1.DS02,DISP=SHR
//STEP3    EXEC PGM=DSNJLOGF
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSUT1   DD DSN=DSNC610.LOGCOPY2.DS01,DISP=SHR
//STEP4    EXEC PGM=DSNJLOGF
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSUT1   DD DSN=DSNC610.LOGCOPY2.DS02,DISP=SHR
```

DSNJLOGF output

```
DSNJ991I DSNJLOGF START OF LOG DATASET PREFORMAT FOR JOB LOGFRMT  STEP1
DSNJ992I DSNJLOGF LOG DATA SET NAME = DSNC610.LOGCOPY1.DS01
DSNJ996I DSNJLOGF LOG PREFORMAT COMPLETED SUCCESSFULLY, 00015000
         RECORDS FORMATTED
```

Chapter 3-3. DSNJU003 (Change Log Inventory)

The DSNJU003 stand-alone utility changes the bootstrap data sets (BSDSs). You can use the utility to:

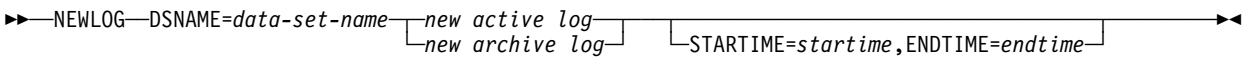
- Add or delete active or archive log data sets
- Add or delete checkpoint records
- Create a conditional restart control record to control the next start of the DB2 subsystem
- Change the VSAM catalog name entry in the BSDS
- Modify the communication record in the BSDS
- Modify the value for the highest-written log RBA value (relative byte address within the log) or the highest-offloaded RBA value

Syntax and options of the control statement

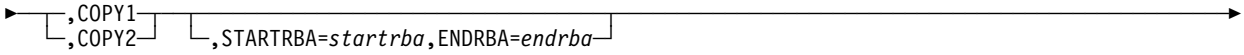
For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.

DSNJU003 (change log inventory) syntax diagram

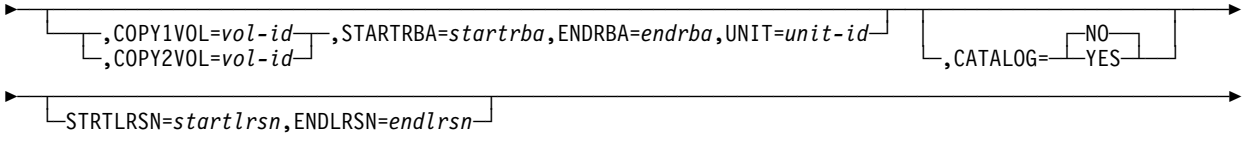
NEWLOG statement



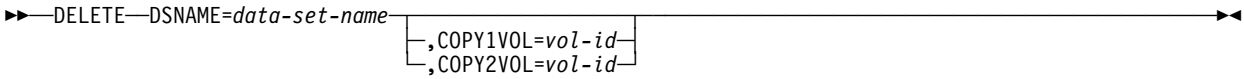
new active log:



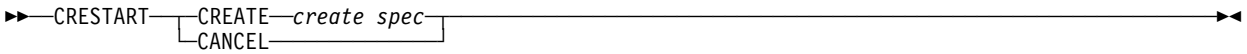
new archive log:



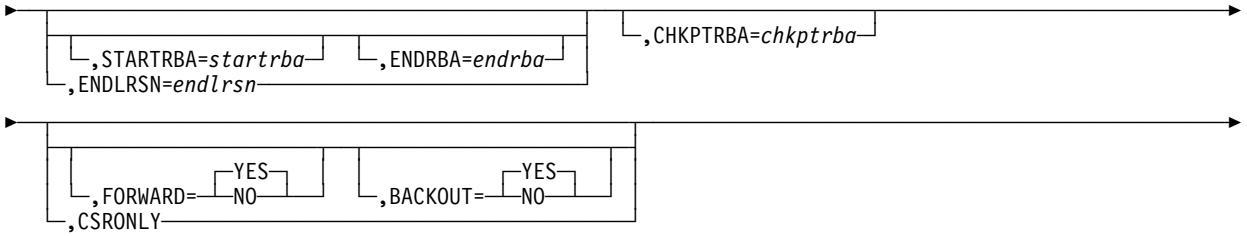
DELETE statement



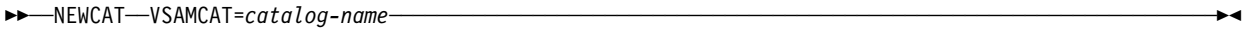
CRESTART statement



create spec:

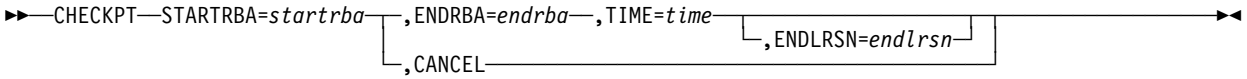


NEWCAT statement

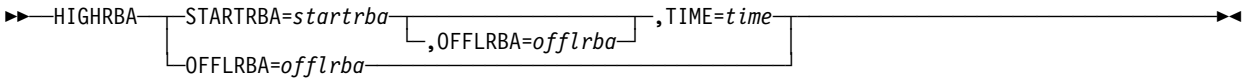


DDF statement

CHECKPT statement



HIGHRBA statement



Option descriptions

For a description of how utility statements are parsed, and how to read a list of option identifiers and specifications like the one that follows, see “Control statement coding rules” on page 407.

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 you specified on install panels DSNTIPA2 and DSNTIPH.

  — *x* is 1 for the first copy of the logs and 2 for the second copy.

  — A*nnnnnnn* represents the series of low-level qualifiers 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 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. 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 specified in the Change Log Inventory NEWLOG statements. For new archive logs 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.

DELETE Deletes all information about the specified log data set or data set volume from the bootstrap data sets.

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.

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 Section 4 (Volume 1) of *DB2 Administration Guide*.

NEWCAT Changes the VSAM catalog name in the BSDS.

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 only include those values 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.

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 Section 4 (Volume 1) of *DB2 Administration Guide*.

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 Section 4 (Volume 1) of *DB2 Administration Guide*.

DSNAME=*data-set-name*
Specifies a log data set.

data-set-name can be up to 44 characters long.

STARTIME=*startime*
Enables you to record the start time of the RBA in the BSDS. This is an optional field. The timestamp format with valid values in parentheses is as follows:

yyyydddhhmmsst

where:

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 must also be specified.

ENDTIME=*endtime*

Enables you to record the end time of the RBA in the BSDS. This is an optional field. For the timestamp format, see the STARTIME option. The ENDTIME value must be greater than or equal to the value of STARTIME.

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*

startrba is a hexadecimal number of up to 12 characters. If you use fewer than 12 characters, leading zeros are added. *startrba* must end with '000' or DB2 returns a DSNJ4381 error message. The RBA can be obtained 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 specified by DSNAME.

On the CRESTART statement, *startrba* is the earliest RBA of the log 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.

for the *startrba* format, see the NEWLOG statement.

On the HIGHRBA statement, *startrba* denotes the log RBA of the highest-written log record in the active log data sets.

For the *startrba* format, see the NEWLOG statement.

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 'FFF' or DB2 will return 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 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 written after restart. Any log information in the bootstrap data set and the active logs with an RBA 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.

For the *endrba* format, see the NEWLOG statement.

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 cataloged.

NO Indicates that the archive log data set is not cataloged. All subsequent allocations of the data set are made using the unit and volume information specified on the statement.

YES Indicates that the archive log data set is cataloged. All subsequent allocations of the data set are made using the catalog.

DB2 requires that all archive log data sets on DASD be cataloged. Select CATALOG=YES if the archive log data set is on DASD.

STRTLRSN=*startlrsn*

On the NEWLOG statement, *startlrsn* is the LRSN in 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.

On the NEWLOG statement, *endlrsn* is the LRSN in log record header of the last log record on the new archive data set.

On the CRESTART statement, *endlrsn* is the LRSN of the last log record to be used during restart. Any log information in the bootstrap data set and the active logs with an LRSN greater than *endlrsn* is discarded. If you omit ENDLRSN, DB2 determines the end of the log range.

The ENDLRSN option is valid only in a data sharing environment. It cannot be specified with STARTRBA or ENDRBA.

On the CHECKPT statement, *endlrsn* is the LRSN of the end checkpoint log record.

CREATE Creates a new conditional restart control record. When the new record is created, the previous control record becomes inactive.

CANCEL **On the CRESTART statement,** CANCEL 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 CHECKPT statement, CANCEL deletes the checkpoint queue entry that contains a starting RBA that matches the parameter specified by the STARTRBA keyword.

CHKPTRBA=*chkptrba*

Is 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 selected by the utility. However, *chkptrba* must be in the range 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 taken.

FORWARD= Indicates whether to use the forward-log-recovery phase of DB2 restart, which reads the log forward 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)

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.

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.

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.

LOCATION=*location-name*

Changes the LOCATION value in the BSDS.

location-name specifies the name of your local DB2 site.

LUNAME=*luname*

Changes the LUNAME value in the BSDS.

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 Specifying a *password* is optional. It 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 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.

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 used by DDF to accept incoming connection requests. This value must be a decimal number between 0 and 65534; 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 used by DDF to accept incoming DRDA two-phase commit resynchronization requests. This value must be a decimal number between 0 and 65534; 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 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.

TIME=*time* **On the CHECKPT statement**, TIME gives the time the start checkpoint record was written.

For timestamp format, see the STARTIME option on the NEWLOG statement on page 414.

On the HIGHRBA statement, TIME specifies when the log record with the highest RBA was written to the log.

For timestamp format, see the STARTIME option on the NEWLOG statement on page 414.

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 `'FFF'`.

Before running DSNJU003

This section contains information you need to be aware of prior to running DSNJU003.

Environment

The utility should be executed only as a batch job when DB2 is not running. It can be executed when DB2 is running, but results can be inconsistent.

Authorization required

The authorization ID of the DSNJU003 job must have the requisite RACF authorization.

Control statement

See “Syntax and options of the control statement” on page 411 for DSNJU003 syntax and option descriptions.

Required and optional data sets

DSNJU003 recognizes DD statements with the following ddnames:

JOBCAT
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 integrated facility 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 changed 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 containing 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 following 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 specified in the statement in error and in any subsequent statements.

Using DSNJU003

This section describes the following tasks associated with running the DSNJU003 utility:

Running DSNJU003

The following statement executes the utility and 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.
3. Use DSNJU003 to register the new data sets in the BSDS.

For example, specify:

```
NEWLOG DSNAME=DSNC610.LOGCOPY1.DS04,COPY1
NEWLOG DSNAME=DSNC610.LOGCOPY2.DS04,COPY2
```

If you are copying 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.

If you are archiving to DASD and the size of your active logs has been increased, you might find it necessary to increase the size of your archive log data sets.

Deleting: To delete information about an active log data set from the BSDS, you might specify:

```
DELETE DSNAME=DSNC610.LOGCOPY1.DS01
DELETE DSNAME=DSNC610.LOGCOPY2.DS01
```

Recording: To record information about an existing active log data set in the BSDS, you might specify:

```
NEWLOG DSNAME=DSNC610.LOGCOPY2.DS05,COPY2,STARTIME=19910212205198,
              ENDTIME=19910412205200,STARTRBA=43F8000,ENDRBA=65F3FFF
```

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 up.
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 have 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. Execute 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 on page 422 through 7 on page 422 to enlarge the active logs that were just archived.

Although it is not necessary for all log data sets to be the same size, from an operational standpoint it is more consistent and efficient. If the log data sets are not the same size, it is more difficult to track your system's logs. Space can be wasted if you are using dual data sets of different sizes because they will fill only to the size of the smallest, not using the remaining space on the larger one.

If you are archiving to DASD and the size of your active logs has been increased, you might find it necessary to increase the size of your archive log data sets. Refer to 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:

```
NEWLOG DSNAME=DSNC610.ARCHLOG1.D89021.T2205197.A0000015,COPY1VOL=DSNV04,
UNIT=TAPE,STARTRBA=3A190000,ENDRBA=3A1F0FFF,CATALOG=NO
```

Deleting: To delete an entire archive log data set from one or more volumes, you might specify:

```
DELETE DSNAME=DSNC610.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 back-out, use a statement similar to this:

```
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 this:

```
CRESTART CREATE,STARTRBA=4A000,ENDRBA=4A000
```

In most cases, when doing a cold start, make sure that the STARTRBA and ENDRBA are set to an RBA value greater than the highest RBA used.

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 have been using 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 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 established at DB2 installation time is:

```
prefix.LOGCOPYn.DSmm
```

where *n*=1 for all primary dog data sets and *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 running 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 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 set themselves do not require deletion and redefinition.

To ensure consistent results, execute the Change Log Inventory utility on the same MVS system where the DB2 online subsystem is executing.

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 DASD volumes containing any DB2 data sets. This will enable a possible fallback. The backup data sets must be generated when DB2 is not active.

Altering references to NEWLOG and DELETE data sets

The NEWLOG and DELETE statements 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.

Specifying the NEWCAT statement

NEWCAT defines the high-level qualifier used for:

- Catalog table spaces and index spaces
- Directory table spaces and index spaces

At startup, the DB2 system checks that the name 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” on page 426 for an additional function of NEWCAT.

When you change the high-level qualifier using the NEWCAT statement, you might specify:

```
//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 running the Change Log Inventory utility with the NEWCAT statement, output similar to the output in Figure 25 is generated.

```
 NEWCAT VSAMCAT=DBP1
SNJ210I OLD VASAM CATALOG NAME=DSNC120 NAME=DBP1
DSNJ225I NEWCAT OPERATION COMPLETED SUCCESSFULLY
DSNJ200I DSNJU003 CHANGE LOG INVENTORY UTILITY
         PROCESSING COMPLETED SUCCESSFULLY
```

Figure 25. Output produced when changing high-level qualifier

Renaming DB2 system data sets

Occasionally, you may 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 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 recreate 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 may 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 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, it can be deleted.

Renaming DB2 archive log data sets

You do not need to rename archive log data sets, because:

- Old archive logs are replaced as a part of the normal maintenance cycle.
- The RECOVER utility works with archive logs containing different high-level qualifiers.

To modify the high-level qualifier for archive log data sets, you need to reassemble DSNZPARM.

Sample control statements

Example 1: Adding a new archive log data set

```
NEWLOG DSNAME=DSNREPAL.A0001187,COPY1VOL=DSNV04,UNIT=SYSDA,
STARTRBA=3A190000,ENDRBA=3A1F0000,CATALOG=NO
```

Example 2: Deleting a data set

```
DELETE DSNAME=DSNREPAL.A0001187,COPY1VOL=DSNV04
```

Example 3: Creating a new conditional restart record. The following statement creates a new conditional restart control record, specifying no backward-log recovery and log truncation (a new relative byte address for the end of the log).

```
CRESTART CREATE,BACKOUT=NO,ENDRBA=000000010000
```

Example 4: Adding a communication record to the BSDS

```
DDF LOCATION=USIBMSTODB22,LUNAME=STL#M08,PASSWORD=$STL@290
```

Chapter 3-4. 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 copy1 and copy2 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

In a data sharing environment, the DSNJU004 utility can list information from any or all BSDSs of a data sharing group.

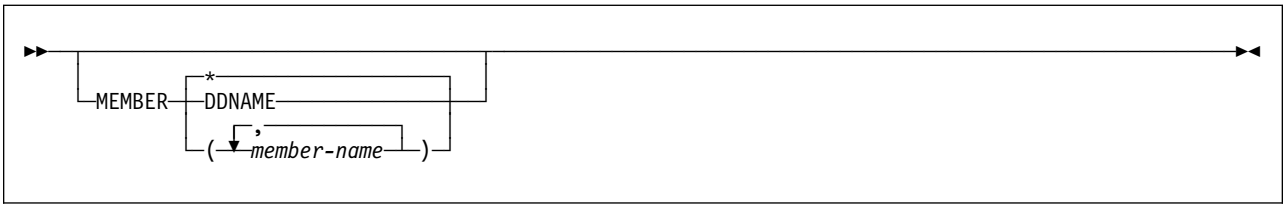
Additional information regarding the DSNJU004 utility appears in Section 4 (Volume 1) of *DB2 Administration Guide*.

Syntax and options of the control statement

For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.

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

This option 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 pointed to by the M*xx*BSDS DD statements.

(*member-name***)**
Prints information for only the group members named.

Before running DSNJU004

This section contains information you need to be aware of prior to running DSNJU004.

Environment

The DSNJU004 program runs as a batch job.

This utility can be executed both 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 MVS 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 429 for DSNJU004 syntax and option descriptions. See “Sample control statement” on page 431 for an example of a control statement.

Required and optional data sets

DSNJU004 recognizes DD statements with the following ddnames:

JOBCAT
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 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. Be sure the BSDS name 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 members required. 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 MVS system where the DB2 online subsystem is executing.
- 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 made by the Change Log Inventory utility.

Sample control statement

The following statement prints information from the BSDS of each member in the data sharing group:

```
//PLM      EXEC PGM=DSNJU004
//SYSUT1   DD DSN=DBD1.BSDS01,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
           MEMBER *
```

DSNJU004 (Print Log Map) output

Figure 26 on page 435 shows an example of the Print Log Map utility output, with the following information:

- The data set name (DSN) of the BSDS.
- The system date and time (SYSTEM TIMESTAMP), which is set at the time the subsystem stops.
- The date and time the BSDS was last changed by the Change Log Inventory utility (listed as the UTILITY TIMESTAMP).
- The integrated catalog facility catalog name associated with the BSDS.

- The highest RBA written. The value is updated each time the log buffers are physically written to DASD.
- The highest RBA 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 437.
- The contents of the checkpoint description queue. For a description of this output, see Figure 27 on page 437.
- The distributed data facility (DDF) communication record. This record contains the DB2-defined location name, the VTAM-defined LUNAME. DB2 uses this information to establish the distributed database environment.

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 26 on page 435 shows an example of the Print Log Map utility output. The following timestamps are included in the header section of the report:

System timestamp Reflects the date and time 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 you have specified and the system activity rate, the BSDS can be updated several times a second, or might not be updated for several seconds, minutes, or even hours.
- When, due to an error, DB2 drops into single-BSDS mode from its normal dual BSDS mode. This can 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 purposely force a timestamp mismatch with the disabled BSDS.

Utility timestamp The date and time 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 report:

Active log date The date the active log data set was originally allocated on the DB2 subsystem.

Active log time The time 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 shown in Figure 28 on page 437:

Conditional restart control record
The current time and date. This data is reported as information only and is not kept in the BSDS.

CRCR created The time and date of creation of the CRCR via the CRESTART option in the Change Log Inventory utility.

Begin restart The time and date the conditional restart was attempted.

End restart The time and date the conditional restart ended.

STARTRBA (timestamp)
The time the control interval was written.

ENDRBA (timestamp)
The time the last control interval was written.

Time of checkpoint The time and date 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 shown in Figure 27 on page 437:

Checkpoint queue The current time and date. This data is reported as information only and is not kept in the BSDS.

Time of checkpoint The time and date the checkpoint was taken.

DDF communication record (heading)
The current time and date. This data is reported as 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 listed in Table 77 on page 434. For an example of how the status appears in Print Log Map utility output, see Figure 26 on page 435.

Table 77. Statuses of active log data sets

Status	Meaning
NEW	The data set has been defined but never used by DB2, or the log was truncated at a point prior to the data set. 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 while truncating the data set following a write I/O error. See Section 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.

```
************************************************************************************************************************
*                                                                                                                      *
*              LOG MAP OF THE BSDS DATA SET BELONGING TO MEMBER 'V61A    ' OF GROUP 'DSNCAT  '.                       *
*                                                                                                                      *
************************************************************************************************************************
RELEASE LEVEL OF BSDS -  ACTIVE=2.3 AND ABOVE   ARCHIVE=2.3 AND ABOVE   DDNAME=SYSUT1
   LOG MAP OF BSDS DATA SET COPY 1, DSN=DSNC610.BSDS01
   LTIME INDICATES LOCAL TIME, ALL OTHER TIMES ARE GMT.
         DATA SHARING MODE IS ON
         SYSTEM TIMESTAMP   - DATE=1995.230  LTIME=15:01:02.70
         UTILITY TIMESTAMP  - DATE=1995.230  LTIME= 9:40:37.02
         VSAM CATALOG NAME=DSNC610
         HIGHEST RBA WRITTEN       0000020648F8  1995.230  22:00:13.0
         HIGHEST RBA OFFLOADED     000000000000
         RBA WHEN CONVERTED TO V4  000001DD9AE8
         MAX RBA FOR TORBA         000001DD9AE8
         MIN RBA FOR TORBA         000000000000
         STCK TO LRSN DELTA        000000000000
       THIS BSDS HAS MEMBER RECORDS FOR THE FOLLOWING MEMBERS:
         HOST MEMBER NAME:            V61A
           MEMBER ID:                 1
           GROUP NAME:                DSNCAT
           BSDS COPY 1 DATA SET NAME: DSNC610.BSDS01
           BSDS COPY 2 DATA SET NAME: DSNC610.BSDS02
         MEMBER NAME:                 V61B
           MEMBER ID:                 2
           GROUP NAME:                DSNCAT
           BSDS COPY 1 DATA SET NAME: DSNC610.BSDS01
           BSDS COPY 2 DATA SET NAME: DSNC610.BSDS02
ACTIVE LOG COPY 1 DATA SETS
  START RBA/LRSN/TIME   END RBA/LRSN/TIME      DATE    LTIME  DATA SET INFORMATION
 --------------------  --------------------  --------  -----  --------------------
   00000189C000           000001C1FFFF       1995.168  16:30  DSN=DSNC610.LOGCOPY1.DS02
     A974FB6CC2FD           A974FBBFD37C     PASSWORD=(NULL)  STATUS=REUSABLE
 1995.171  09:26:32.1  1995.171  09:27:59.2
   000001C20000           000001DD9FFF       1995.168  16:30  DSN=DSNC610.LOGCOPY1.DS03
     A974FBC0D182           A994BA682B38     PASSWORD=(NULL)  STATUS=TRUNCATED, REUSABLE
 1995.171  09:28:00.2  1995.196  15:26:02.2
   000001DDA000           00000215DFFF       1995.168  16:30  DSN=DSNC610.LOGCOPY1.DS01
     A994BA682B39           ............     PASSWORD=(NULL)  STATUS=REUSABLE
 1995.196  15:26:02.2  ........  ..........
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
 --------------------  --------------------  --------  -----  --------------------
   00000189C000           000001C1FFFF       1995.168  16:30  DSN=DSNC610.LOGCOPY2.DS02
     A974FB6CC2FD           A974FBBFD37C                     STATUS=REUSABLE
 1995.171  09:26:32.1  1995.171  09:27:59.2
   000001C20000           000001DD9FFF       1995.168  16:30  DSN=DSNC610.LOGCOPY2.DS03
     A974FBC0D182           A994BA682B38                     STATUS=TRUNCATED, REUSABLE
 1995.171  09:28:00.2  1995.196  15:26:02.2
   000001DDA000           00000215DFFF       1995.168  16:30  DSN=DSNC610.LOGCOPY2.DS01
     A994BA682B39           ............                     STATUS=REUSABLE
 1995.196  15:26:02.2  ........  ..........
ARCHIVE LOG COPY 2 DATA SETS
NO ARCHIVE DATA SETS DEFINED FOR THIS COPY
DSNJ401I DSNRJPCR RESTART CONTROL RECORD NOT FOUND
```

Figure 26 (Part 1 of 2). Sample print log map utility output

```
                        CHECKPOINT QUEUE
                      22:02:23 AUGUST 18, 1995
      TIME OF CHECKPOINT       22:01:41 AUGUST 18, 1995
      BEGIN CHECKPOINT RBA             000002065090
      END CHECKPOINT RBA               000002066C9A
      TIME OF CHECKPOINT       22:00:10 AUGUST 18, 1995
      BEGIN CHECKPOINT RBA             000002062D08
      END CHECKPOINT RBA               0000020648F8
      SHUTDOWN CHECKPOINT
      TIME OF CHECKPOINT       21:50:48 AUGUST 18, 1995
      BEGIN CHECKPOINT RBA             000002061090
      END CHECKPOINT RBA               000002062C9A
      TIME OF CHECKPOINT       21:19:46 AUGUST 18, 1995
      BEGIN CHECKPOINT RBA             00000205ED08
      END CHECKPOINT RBA               0000020608F8
      SHUTDOWN CHECKPOINT

⋮
      TIME OF CHECKPOINT       23:41:41 JUNE 17, 1995
      BEGIN CHECKPOINT RBA             000000074F30
      END CHECKPOINT RBA               000000079A42
      TIME OF CHECKPOINT       23:41:15 JUNE 17, 1995
      BEGIN CHECKPOINT RBA             000000035000
      END CHECKPOINT RBA               000000039EBC
      TIME OF CHECKPOINT       23:34:35 JUNE 17, 1995
      BEGIN CHECKPOINT RBA             0000000000BA
      END CHECKPOINT RBA               000000001C1E
DSNJ401I DSNJU104 ARCHIVE LOG COMMAND HISTORY RECORD NOT FOUND
                 **** DISTRIBUTED DATA FACILITY ****
                        COMMUNICATION RECORD
                      22:02:23 AUGUST 18, 1995
LOCATION=SANTA_TERESA_LAB  LUNAME=LUND0  PASSWORD=(D02DN)
DSNJ200I DSNJU004 PRINT LOG UTILITY PROCESSING COMPLETED SUCCESSFULLY
```

Figure 26 (Part 2 of 2). Sample print log map utility output

The ARCHIVE LOG COMMAND HISTORY in the output above was created as follows:

- The first entry in the history was created by issuing the ARCHIVE LOG command:

  ```
  -ARCHIVE LOG MODE(QUIESCE) WAIT(YES) TIME(999)
  ```

- The next entry was created by issuing the ARCHIVE LOG command without a time parameter. The D after the time signifies that the default DSNZPARM TIME value (3 seconds) was used.

  ```
  -ARCHIVE LOG MODE(QUIESCE)
  ```

- The last two entries in the history were created by issuing the ARCHIVE LOG command as follows:

  ```
  -ARCHIVE LOG
  ```

- The values in the TIME column of the ARCHIVE LOG COMMAND HISTORY section of the report represent the time the ARCHIVE LOG command was issued. This time value is saved in the BSDS and is converted to printable format at the time 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 description. A sample description of a checkpoint record in the queue is shown in Figure 27.

```
                   CHECKPOINT QUEUE
                 13:02:50 MAY 14, 1999
TIME OF CHECKPOINT       13:35:20 MAY 14, 1999
BEGIN CHECKPOINT RBA             000000047C10
END CHECKPOINT RBA               000000048510
TIME OF CHECKPOINT       13:21:49 MAY 14, 1999
BEGIN CHECKPOINT RBA             000000035010
END CHECKPOINT RBA               000000035780
TIME OF CHECKPOINT       13:01:26 MAY 14, 1999
BEGIN CHECKPOINT RBA             000000029000
END CHECKPOINT RBA               0000000297A0
```

Figure 27. Sample print log map description of checkpoints

A sample description of a conditional restart control record is shown in Figure 28.

```
CRCR IDENTIFIER  0001
     USE COUNT   1
     RECORD STATUS
         CRCR NOT ACTIVE
         SUCCESSFUL RESTART
     PROCESSING STATUS
         FORWARD = YES
         BACKOUT = NO
     STARTRBA                          000000028894
     ENDRBA                            000000058000
     EARLIEST REQUESTED RBA            000000027B00
     FIRST LOG RECORD RBA              0000000288B0
     ORIGINAL CHECKPOINT RBA           00000005A390
     NEW CHECKPOINT RBA (CHKPTRBA)     000000047C10
     END CHECKPOINT RBA                000000048510
     CRCR CREATED              11:21:08 MAY 14, 1999
     BEGIN RESTART             11:26:26 MAY 14, 1999
     END RESTART               11:31:38 MAY 14, 1999
     TIME OF CHECKPOINT        10:34:31 MAY 14, 1999
     RESTART PROGRESS                    STARTED     ENDED
                                         =======     =====
         CURRENT STATUS REBUILD            YES         YES
         FORWARD RECOVERY PHASE            YES         YES
         BACKOUT RECOVERY PHASE            YES         YES
```

Figure 28. Sample print log map description of a CRCR

Chapter 3-5. DSN1CHKR

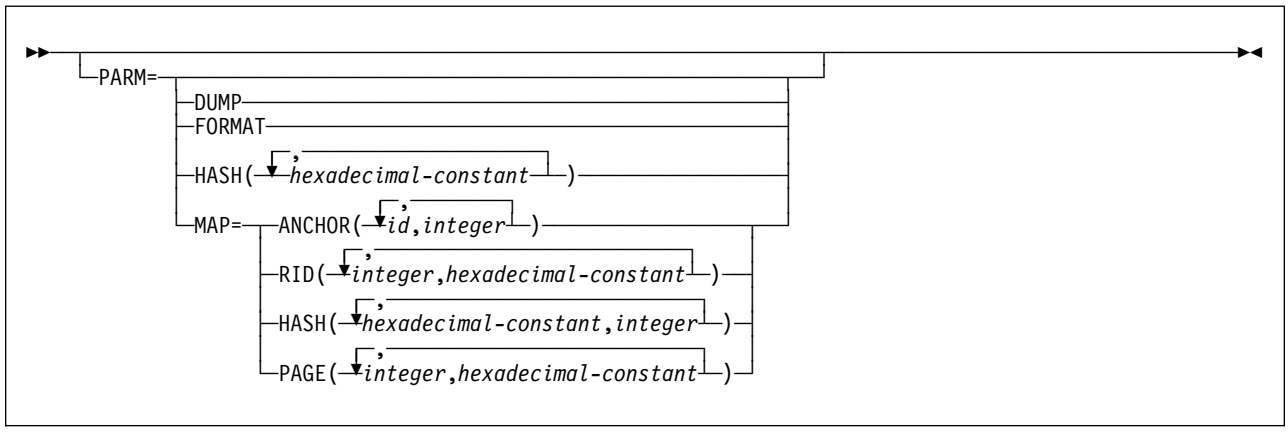
The DSN1CHKR utility verifies the integrity of DB2 directory and catalog table spaces. DSN1CHKR scans the specified table space for broken links, broken hash chains, and records that are not part of any link or chain.

Use DSN1CHKR on a regular basis to promptly detect any damage to the catalog and directory.

Syntax and options of the control statement

For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.

DSN1CHKR syntax diagram



Option descriptions

The following parameters are optional. Parameters should be specified on the EXEC card and can be specified in any order. 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 in dump format. If you specify DUMP, you cannot specify the FORMAT parameter.

FORMAT Specifies that printed table space pages, if any, are 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 and 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 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 it is broken by running DSN1CHKR without any parameters, or with FORMAT or DUMP only.

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 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 maps.

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 starts mapping.

integer is the page and record, in the form *pppppprr*, where *pppppp* is the page number and *rr* is the record number.

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 hashes and maps 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 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

This section contains information you need to know before you run DSN1CHKR.

DSN1CHKR is a diagnosis tool; it executes outside the control of DB2. Detailed knowledge of DB2 data structures is required to make proper use of this service aid.

Environment

Run the DSN1CHKR program as an MVS job.

You must not run DSN1CHKR on a table space while it is active under DB2. Ensure that no database operations are performed while DSN1CHKR runs by issuing the STOP DATABASE command for the database and table space you want to check.

Authorization required

None is required. However, if any of the data sets is RACF protected, the authorization ID of the job must have the necessary RACF authority.

Control statement

See “Syntax and options of the control statement” on page 439 for DSN1CHKR syntax and option descriptions.

Required data sets: DSN1CHKR uses two DD cards. Specify the data set for the utility's output with the DD card SYSPRINT. Specify the first data set piece of the table space that is to be checked with the DD card SYSUT1.

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 of the DB2 data set created by the DSN1COPY utility. Disposition for this data set must be specified as DISP=OLD to ensure that it is not in use by DB2. Disposition for this data set must be specified as DISP=SHR only when the table space you want to check has been stopped by the STOP DATABASE command.

Restrictions

This section contains restrictions to be aware of before running DSN1COMP.

Running DSN1COPY before DSN1CHKR

DSN1CHKR requires a VSAM data set as input; it cannot check a physical sequential data set.

Full image copies created with the COPY utility cannot be used directly as input to DSN1CHKR. If the image copy is created with SHRLEVEL REFERENCE and is a full image copy, you can copy it into a VSAM data set with DSN1COPY and check it with DSN1CHKR.

Full image copies created with DFSMS concurrent copy cannot be used by DSN1CHKR. The file format is incompatible with DSN1COPY, so the DFSMS Concurrent Copy IC data set cannot be copied to a VSAM data set.

Recommendation: First copy the stopped table space to a temporary data set 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 a valid table space.

Do not run DSN1CHKR on the following table spaces:

- DSNDB06.SYSCOPY
- DSNDB06.SYSDDF
- DSNDB06.SYSGPAUT
- DSNDB06.SYSPKAGE
- DSNDB06.SYSSTATS
- DSNDB06.SYSSTR
- DSNDB06.SYSUSER
- DSNDB01.SCT02
- DSNDB01.SPT01
- DSNDB01.SYSLGRNX
- DSNDB01.SYSUTILX

Sample control statements

Example 1: Running DSN1CHKR on a temporary data set. STEP1 allocates a temporary data set. STEP2 stops database DSNDB06 with the STOP DATABASE command. STEP3 copies the target table space into the temporary data set with DSN1COPY. The CHECK option is used to check the table space for page integrity errors. After DSN1COPY with the check option has ensured that no errors exist, STEP4 restarts the table space for access to DB2 again. STEP5 runs DSN1CHKR on the temporary data set.

DSN1CHKR prints the chains beginning at the pointers specified on the RID option of the MAP parameter. In this example, the first pointer is located on page 2, at an offset of 6 bytes from record 1, and the second pointer is located on page B, at an offset of 6 bytes from record 1.

The RIDs in STEP5 of the example are for example purposes only. Using them results in a error message. Change them to the actual RIDs to be checked.

```
//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(V61A)
     -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 29 (Part 1 of 2). Sample JCL for running DSN1CHKR on a temporary data set

```
//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(V61A)
     -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 29 (Part 2 of 2). Sample JCL for running DSN1CHKR on a temporary data set

Example 2: Running DSN1CHKR on an actual table space. STEP1 stops database DSNDB06 with the STOP DATABASE command. STEP2 runs DSN1CHKR on the target table space; its output 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(V61A)
     -STOP DB(DSNDB06) SPACENAM(SYSDBASE)
 END
/*
```

Figure 30 (Part 1 of 2). Sample JCL for running DSN1CHKR on a stopped table space.

```
//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(V61A)
     -START DB(DSNDB06) SPACENAM(SYSDBASE)
 END
/*
```

Figure 30 (Part 2 of 2). Sample JCL for running DSN1CHKR on a stopped table space.

DSN1CHKR output

Interpreting 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.

Chapter 3-6. DSN1COMP

DSN1COMP estimates space savings to be achieved by DB2 data compression in table spaces. For more information regarding ESA data compression, see Section 2 (Volume 1) of *DB2 Administration Guide*.

This utility can be run on the following types of data sets containing uncompressed data:

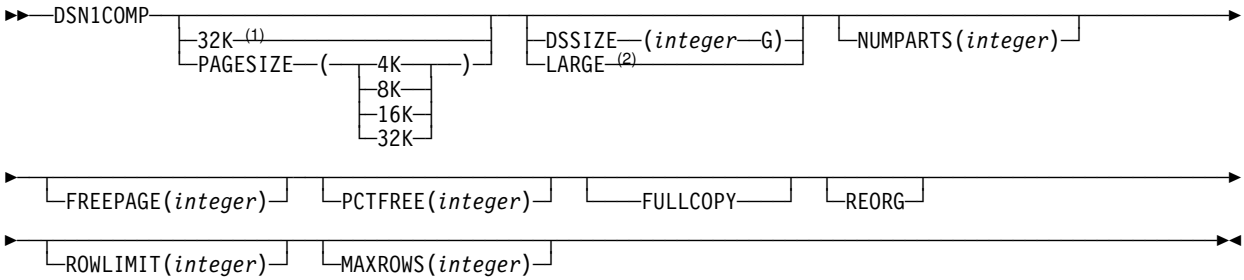
- DB2 full image copy data sets
- VSAM data sets that contain DB2 table spaces
- Sequential data sets that contain DB2 table spaces (for example, DSN1COPY output)

DSN1COMP does not estimate savings for data sets that contain LOB table spaces or index spaces.

Syntax and options of the control statement

For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.

DSN1COMP syntax diagram



Notes:

1 The preferred option is PAGESIZE(32K).

2 The preferred method of specifying LARGE is DSSIZE(4G).

Important: If you specify more than one parameter:

- Separate them by commas (no blanks).
- Specify them in any order.

Option descriptions

Specify one or more of the parameters listed below on the EXEC card to run DSN1COMP.

32K Specifies that the input data set, SYSUT1, has a 32-KB page size. If the SYSUT1 data set has a 32-KB page size, and you do not specify this option, DSN1COMP produces unpredictable results, because the default page size is 4 KB.

The preferred option is **PAGESIZE(32K)**.

PAGESIZE Specifies the page size of the input data set that is defined by SYSUT1. If you specify an incorrect page size, DSN1COMP may 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 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 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 table space that was defined with the LARGE option. If you specify LARGE, then DB2 assumes that the data set has a 4-GB boundary.

The preferred 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 associated with the input data set. Valid specifications range from 1 to 254. 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 may produce unpredictable results.

DSN1COMP terminates and issues message DSN1946I when it encounters an image copy containing 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 pages saved. You must specify an integer in the range 0 to 255. If you specify 0, no pages are included as free space when reporting 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 DFSMS concurrent copy) of your data is 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 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 99,000,000.

Use this option to limit the elapsed time and processor time that DSN1COMP requires. An analysis of the first 5-10 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 5-10 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

This section contains information to keep in mind before you run DSN1COMP.

Environment

Run DSN1COMP as an MVS job.

You can run DSN1COMP even when the DB2 subsystem is not operational. If you choose to use DSN1COMP when the DB2 subsystem is operational, issue the DB2 STOP DATABASE command to be sure that the DB2 data sets that are to be used are not currently allocated to DB2.

DSN1COMP is not meant to be run on table spaces in DSNDB01, DSNDB06, or DSNDB07.

Authorization required

None is required. However, if any of the data sets is RACF-protected, the authorization ID of the job must have RACF authority.

Control statement

See “Syntax and options of the control statement” on page 447 for DSN1COMP syntax and option descriptions.

Required data sets: DSN1COMP uses the DD cards described below:

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.

Disposition for this data set must be specified as OLD (DISP=OLD) to ensure that it is not in use by DB2. Disposition for this data set must be specified as SHR (DISP=SHR) only in circumstances where the DB2 STOP DATABASE command does not work.

The requested operation takes place only for the data set specified. If the input data set belongs to a linear table space or index space that is larger than 2 gigabytes, or if it is a partitioned table space or index space, you must ensure that the correct data set is specified.

Recommendation

This section contains a recommendation for running DSN1COMP.

Determining page size and DSSIZE: Before using DSN1COMP, be sure 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:

```
SELECT  T.CREATOR,
        T.NAME,
        S.PGSIZE,
        CASE S.DSSIZE
          WHEN 0 THEN
            CASE S.TYPE
              WHEN ' ' THEN 2097152
              WHEN 'I' THEN 2097152
              WHEN 'L' THEN 4194304
              WHEN 'K' THEN 4194304
              ELSE NULL
            END
          ELSE S.DSSIZE
        END
  FROM  SYSIBM.SYSTABLES  T,
        SYSIBM.SYSTABLESPACE S
 WHERE  T.DBNAME=S.DBNAME
   AND  T.TSNAME=S.NAME;
```

Using DSN1COMP

This section describes the following tasks associated with running the DSN1COMP utility:

Estimating compression savings achieved by REORG

If you run DSN1COMP with the REORG option on small data sets or specify a small number (*n*) for the ROWLIMIT keyword, the estimates produced might vary greatly from the estimates produced without the REORG option (the default invocation).

Without the REORG option, DSN1COMP uses the first *n* rows to fill the compression dictionary. The remaining rows are processed to provide the compression estimate. Therefore, if the number of rows used to build the dictionary is a significant percentage of the total number of rows in the data set, very little savings will result. With the REORG option, DSN1COMP processes all the rows, including those used to build the dictionary, which produces a greater compression savings estimate.

Including free space in compression calculations

In the DSN1COMP utility's compression estimates, the PCTFREE and FREEPAGE options you specify are taken into consideration. So, if you run the utility using different PCTFREE or FREEPAGE values than the input table space was created with, you get a different value for **noncmppages** in the DSN1940I message than the actual number of pages in the input table space.

Running DSN1COMP on a table space with identical data

If you run DSN1COMP on a table space in which the data is the same for all rows, message DSN1941I is issued, and DSN1COMP does not compute any statistics.

Sample control statements

Example 1: Running DSN1COMP

```
//jobname JOB acct info
//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: Running DSN1COMP using the PCTFREE and FREEPAGE options

```
//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=DSNC610.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=DSNC610.DSNDBD.DB254SP4.TS254SP4.I0001.A001,DISP=SHR
/*
//
```

DSN1COMP output

This section contains sample output 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 31 on page 453 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 31. Sample DSN1COMP report

Chapter 3-7. 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

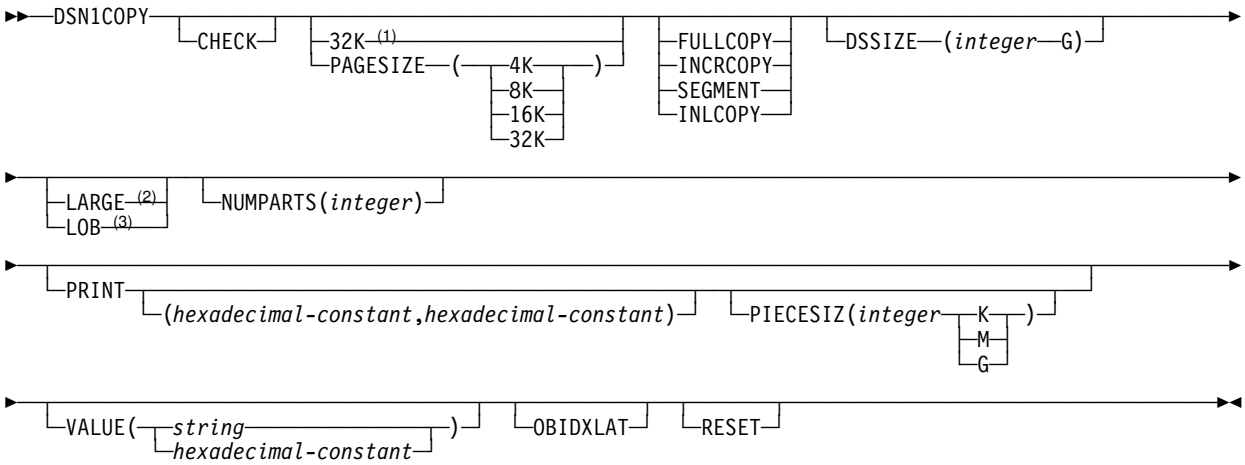
Using DSN1COPY, you can also print hexadecimal dumps of DB2 data sets and databases, check the validity of data or index pages (including dictionary pages for compressed data), translate database object identifiers (OBIDs) to enable moving data sets between different systems, and reset to 0 the log RBA that is recorded in each index page or data page.

DSN1COPY is compatible with LOB table spaces, when you specify the LOB keyword, and omit the SEGMENT and INLCOPY keywords.

Syntax and options of the control statement

For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.

DSN1COPY syntax diagram



Notes:

1 The preferred option is PAGESIZE(32K).

2 The preferred method of specifying LARGE is DSSIZE(4G).

3 You cannot specify the SEGMENT and INLCOPY keywords with the LOB keyword.

Important: If you specify more than one parameter:

- Separate them by commas (no blanks).
- Specify them in any order.

Option descriptions

Specify one or more of the parameters listed below on the EXEC card to run DSN1COPY.

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.

32K Specifies that the SYSUT1 data set has a 32-KB page size. If the SYSUT1 data set has a 32-KB page size, and you do not specify this option, DSN1COPY may produce unpredictable results, because the default page size is 4 KB.

The preferred option is **PAGESIZE(32K)**.

PAGESIZE Specifies the page size of the input data set that is defined by SYSUT1. If you specify an incorrect page size, DSN1COPY may produce unpredictable results.

If you omit PAGESIZE, DSN1COPY tries to determine the page size from the input data set. 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 DFSMS 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 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 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. Zeroed pages 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 DSSIZE, DB2 assumes that the input data set size is 2 GB unless the input data set is a LOB, in which case DB2 assumes a 4 GB input data set size.

integer must match the DSSIZE value 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.

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 preferred 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.

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.

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 terminates.

NUMPARTS(*integer***)**
Specifies the total number of partitions associated with the data set you are using as input or whose page range you are printing.

integer can range from 1 to 254.

DSN1COPY uses this value to calculate the size of its output data sets and to help locate the first page in a range to be printed. If you omit NUMPARTS or specify it as 0, DSN1COPY assumes that your

input file is not partitioned. If you specify a number greater than 64, DSN1COPY assumes 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 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.

PIECESIZ(*integer***)**

Specifies the maximum piece size (data set size) for non-partitioned indexes. The value you specify must match the value specified when the nonpartitioning index was created or altered.

The defaults for PIECESIZ are 2G (2 GB) for indexes backed by non-large table spaces and 4G (4 GB) for indexes 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 option is required for DSN1COPY.

The subsequent keyword K, M, or G, indicates the units of the value 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 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. You must specify two single quotation mark characters before and after the hexadecimal character string.

If you want to search your input file for the string '12345', your JCL should look like this:

```
//STEP1 EXEC PGM=DSN1COPY,PARM='VALUE(12345)'
```

If you want to search for the equivalent hexadecimal character string, your JCL should look like this:

```
//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 cards. DSN1COPY can only translate up to 500 record OBIDs at a time. 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 to be reset to 0. If you specify this option, CHECK processing is performed regardless of whether you specify the CHECK option.

You must use RESET when the output file is used to build a DB2 table space 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 may result in abend reason code 00C2010D when the table space is accessed. For more information about down-level detection, see Section 4 (Volume 1) of *DB2 Administration Guide*.

Before running DSN1COPY

This section contains information to keep in mind before you run DSN1COPY.

Attention: DSN1COPY is not intended to be used in place of the COPY utility or standard backup and recovery procedures. Improper use of DSN1COPY can result in unrecoverable damage and loss of data.

Environment

Execute DSN1COPY as an MVS job, when the DB2 subsystem is either active or not active.

If you execute DSN1COPY when DB2 is active, follow the procedure below:

1. Start the table space as read-only 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 SYSUT1 DD card.
4. Start the table space as read-write using -START DATABASE to return to normal operations.

Authorization required

None is required. However, if any of the data sets is RACF-protected, the authorization ID of the job must have the necessary RACF authority.

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 previously created by DSN1COPY

SYSUT1 should be defined with DISP=OLD to ensure that it is exclusively used by DSN1COPY. If SYSUT1 is a table space or index space, use the following procedure before running DSN1COPY:

1. Issue the following command to determine if the object is stopped:

   ```
   -DISPLAY DATABASE (database_name) SPACENAM(space_name) RESTRICT
   ```

2. If the object is not stopped, issue the following command to stop the object:

   ```
   -STOP DATABASE (database_name) SPACENAME(space_name)
   ```

Only one input DSN1COPY data set is allowed. Concatenated input data sets are not permitted. For a table space consisting of multiple data sets, ensure that you specify the correct data set. For example, if you specify the CHECK option to validate the pages of the second partition of a partitioned table space, code the second data set of the table space for SYSUT1.

Control statement

See “Syntax and options of the control statement” on page 455 for DSN1COPY syntax and option descriptions.

Required data sets

DSN1COPY uses the data sets described below:

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 several DD cards. They are:

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 created by the DSN1COPY or COPY utilities, or a VSAM data set. DSN1COPY assumes that the block size is 4096 bytes (the standard for DB2 data sets).

Disposition for this data set must be specified as DISP=OLD to ensure that it is not in use by DB2. Disposition for this data set must be specified 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 partitioning 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 in the group of VSAM data sets comprising the table space (in other words, DSN=...A002).

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 INCRCOPY is specified. If your output data sets are not defined REUSE, you must use access method services to redefine all the VSAM output data sets you are restoring before you run DSN1COPY. Be sure that any output VSAM data sets are empty (newly defined or REUSE) before running this program.

You might want to specify a DUMMY SYSUT2 DD card if you are doing only page checking or page dumping.

To enable DB2 to obtain necessary information from the integrated catalog facility catalog, do not code unit and volume serial parameters when using VSAM data sets.

SYSXLAT Defines the DBIDs, PSIDs, and OBIDs (ISOBIDs for indexes) to be translated.

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 removes any records that have been previously dropped from the table space.

Each record in the SYSXLAT file must contain a pair of decimal integers and be separated by a nonnumeric character. 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 and target DBIDs; these values can range from -32767 to 65535. The second record contains the source and target PSIDs or ISOBIDs for indexes; these 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 can look like this:

```
260,280
2,10
3,55
6,56
7,57
```

To obtain the names, DBIDs, PSIDs, ISOBIDs, and OBIDs of the tables and indexes needed to create the SYSXLAT file, run the DSNTEP2 sample application on both the source and target systems. The following SQL statements yield the above information:

Note: The example for indexes yields the above information, along with an additional column of data.

Product-sensitive Programming Interface

For table spaces:

```
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:

```
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 are identified below:

- 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 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 card if you are using DSN1COPY for page checking or page dumping. Except when an incremental image copy is being applied (the INCRCOPY parameter), the DB2 table spaces and index spaces either must be empty or must have been defined with the VSAM REUSE parameter. STOGROUP-defined table spaces and index spaces have the REUSE attribute.

Naming the output data set: For your output data set to be useful, you must make sure that it has the same name as the data set you are resetting. You can do this in one of two ways:

- 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 your existing data set was defined 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 the reset operation has been completed, delete the data set you specified as SYSUT1, and rename the SYSUT2 data set, giving it the name of the data set that you just deleted.

If you are using 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 should be the name of the first data set of the table space. DSN1COPY determines the correct target data set. Specify the NUMPARTS parameter to identify the number of partitions in the whole table space.
- **If SYSUT1 is an image copy of a whole partitioned table space**, SYSUT2 should be the name of the first data set of the table space. DSN1COPY allocates all of the target data sets. However, the target data sets must be previously defined 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 single data set of a nonpartitioned table space**, 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 of a multiple data set linear table space**, SYSUT2 should be the name of the first data set of the table space. DSN1COPY allocates all target data sets. However, the target data sets must be previously defined using IDCAMS.

Adding additional volumes for SYSUT2: When a table space or index space is created using STOGROUP, the integrated catalog facility catalog entry has only one volume in the volume list. If the amount of data being restored by DSN1COPY requires more than one volume for SYSUT2, 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 normal VSAM extension. If you want the data set to use the secondary size on the candidate volumes:

1. Run DSN1COPY.
2. Run REORG, or make a full image copy and recover the table space.

This resets the data set and causes normal extensions through DB2.

Restrictions

This section contains restrictions for running DSN1COPY.

You cannot use DSN1COPY to alter data set structure; for example, you cannot 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 data definitions for the source and target table spaces, tables, and indexes are identical; otherwise, unpredictable results can occur.

You cannot use DSN1COPY to 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 this utility to recover log data sets, DSN1COPY abends.

Recommendations

This section contains recommendations for 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 499.

Determining page size and DSSIZE

Before using DSN1COPY, be sure 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:

```
SELECT I.CREATOR,
      I.NAME,
      S.PGSIZE,
      CASE S.DSSIZE
        WHEN 0 THEN CASE S.TYPE
                      WHEN ' ' THEN 2097152
                      WHEN 'I' THEN 2097152
                      WHEN 'L' THEN 4194304
                      WHEN 'K' THEN 4194304
                      ELSE NULL
                    END
        ELSE S.DSSIZE
      END
 FROM SYSIBM.SYSINDEXES I,
      SYSIBM.SYSTABLES  T,
      SYSIBM.SYSTABLESPACE S
 WHERE I.CREATOR='DSN8610' AND
       I.NAME='XEMP1' AND
       I.TBCREATOR=T.CREATOR AND
       I.TBNAME=T.NAME AND
       T.DBNAME=S.DBNAME AND
       T.TSNAME=S.NAME;
```

Using DSN1COPY

This section describes the following tasks associated with running the DSN1COPY utility:

Altering a table before running DSN1COPY

If you do an ALTER TABLE ADD COLUMN, only the description of the table changes. You must run REORG on the table space (so the data matches its description) before you 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 performs validity checking on one page at a time.

You must run a CHECK utility job on the table space 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.

Translating DB2 internal identifiers

| If you use DSN1COPY to load data into a table space or index and you do not specify the OBIDXLAT parameter, be careful not to invalidate DB2 internal identifiers (like object descriptors or OBIDs) that are embedded within the data. Those OBIDs can become invalid in the following ways:

- When tables are dropped and recreated after the data DSN1COPY saved was created and before it was used
- 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 that all table spaces, indexes, and tables are defined or dropped in the source and target databases

To protect against invalidating the OBIDs, specify the OBIDXLAT parameter of DSN1COPY. This performs OBID, DBID, or PSID translation before the data is copied.

Using an image copy as input to DSN1COPY

If you want to include the FULLCOPY parameter in order to use image copies as input to DSN1COPY, be sure that those image copies are produced using the COPY utility with the SHRLEVEL REFERENCE parameter. Using this parameter ensures that the data 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

The RESET option resets to 0 the log RBAs that are recorded in a table space or index space. DSN1COPY performs CHECK processing whether it is explicitly requested or not.

Do not specify the RESET parameter for page sets that are in group buffer pool RECOVER-pending (GRECP) status.

Copying multiple data set table spaces

When using DSN1COPY to copy from an image copy of an individual data set or all data sets of a multiple data set table space to a table space data set, specify the following SYSUT2 data sets:

- **If SYSUT1 is an image of a single partition**, SYSUT2 should be the name of the first data set of the table space. DSN1COPY determines the correct target data set. Code the NUMPARTS(*nn*) parameter, where *nn* is the number of partitions in the whole table space.
- **If SYSUT1 is an image copy of a whole partitioned table space**, SYSUT2 should be the name of the first data set of the table space. In this case, DSN1COPY allocates all of the target data sets. However, the target data sets must be previously defined using IDCAMS. Code the NUMPARTS(*nn*) parameter, where *nn* is the number of partitions in the whole table space.
- **If SYSUT1 is an image copy of a single data set of a multiple data set linear (non-partitioned) table space**, SYSUT2 should be the name of the actual output data set. Do not specify NUMPARTS, because this parameter is only for partitioned table spaces.
- **If SYSUT1 is an image copy of all data sets of a multiple data set linear table space**, SYSUT2 should be the name of the first data set of the table space. DSN1COPY allocates all target data sets.

Restoring indexes with DSN1COPY

When a table space has been restored to an earlier point using either the TOCOPY option of RECOVER or the DSN1COPY utility, restore the indexes in one of three ways:

- Use the RECOVER utility if the index was defined with COPY YES and you have a full image copy available.
- An alternative is to 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 have been copied at the same time as the data; otherwise, inconsistencies may exist.
- If you don't have an image copy of the index available, a safe option is to use the REBUILD INDEX utility, which will reconstruct the indexes from the data. However, for table spaces with millions of rows, REBUILD INDEX might take a

long time. For more information about the REBUILD INDEX utility, refer to “Chapter 2-13. REBUILD INDEX” on page 209.

Restoring table spaces with DSN1COPY

It is not possible to use RECOVER TOCOPY to apply an image copy data set that is not referred to in a SYSIBM.SYSCOPY row for that table space or data set. An attempt to do so results in the message `"TOCOPY DATASET NOT FOUND"`.

The SYSIBM.SYSCOPY row might have been removed by the MODIFY utility. If this has happened, and the image copy is a full image copy with SHRLEVEL REFERENCE, you can restore the table space or data set with DSN1COPY.

It is also possible to restore to an incremental image copy with DSN1COPY but, to ensure data integrity, you need to have first restored the previous full image copy and any intermediate incremental image copies. It is your responsibility to get the sequence of image copies right. DB2 cannot help ensure the proper sequence.

If you use DSN1COPY for point-in-time recovery, the table space becomes not recoverable. Because DSN1COPY executed outside of DB2's control, DB2 is not aware that you recovered to a point-in-time. To ensure recoverability of the affected table space after point-in-time recovery using DSN1COPY:

1. Clean out old image copies, with MODIFY AGE(*).
2. Create one or more full image copies with SHRLEVEL REFERENCE.

Printing with DSN1COPY

If you want to print one or more pages without having the 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 data set specified. If the input data set belongs to a linear table space or index space that is larger than 2 gigabytes or if it is a partitioned table space or partitioning index, you must ensure that you specify 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 in the group of VSAM data sets comprising the table space (in other words, DSN=...A002).

To print a full image copy data set (rather than recovering a table space), specify a DUMMY SYSUT2 DD card and specify the FULLCOPY parameter.

Copying tables from one subsystem to another

Special care must be taken when copying a table containing 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 corresponding to the identity column for this table on the source subsystem. Add the INCREMENT value to the MAXASSIGNEDVAL value to determine the next value to be assigned (*nv*).
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 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 control statements

| ***Example 1: Running DSN1COPY with the CHECK option***

```
//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 DB2 internal identifiers using the OBIDXLAT*** | ***parameter***

```
//EXECUTE EXEC PGM=DSN1COPY,PARM='OBIDXLAT'
//STEPLIB  DD  DSN=PDS CONTAINING DSN1COPY
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  DSN=DSNC610.DSNDBC.DSN8D61P.DSN8S61C.I0001.A001,
//             DISP=OLD
//SYSUT2   DD  DSN=DSNC618.DSNDBC.DSN8D61P.DSN8S61C.I0001.A001,
//             DISP=OLD
//SYSXLAT  DD  *
260,280
2,10
3,55
6,56
7,57
/*
```

| ***Example 3: Printing a single page of a partitioned table space***

```
//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

```
//PRINT2 EXEC PGM=DSN1COPY,PARM=(PRINT(F0000,F000F),PIECESIZ(64M))
//* PRINT 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
```

DSN1COPY output

Interpreting 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.

Chapter 3-8. 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 Support Center personnel analyze the log in detail. (This book does not include a full description of the detail report.)
- A *summary report* 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:

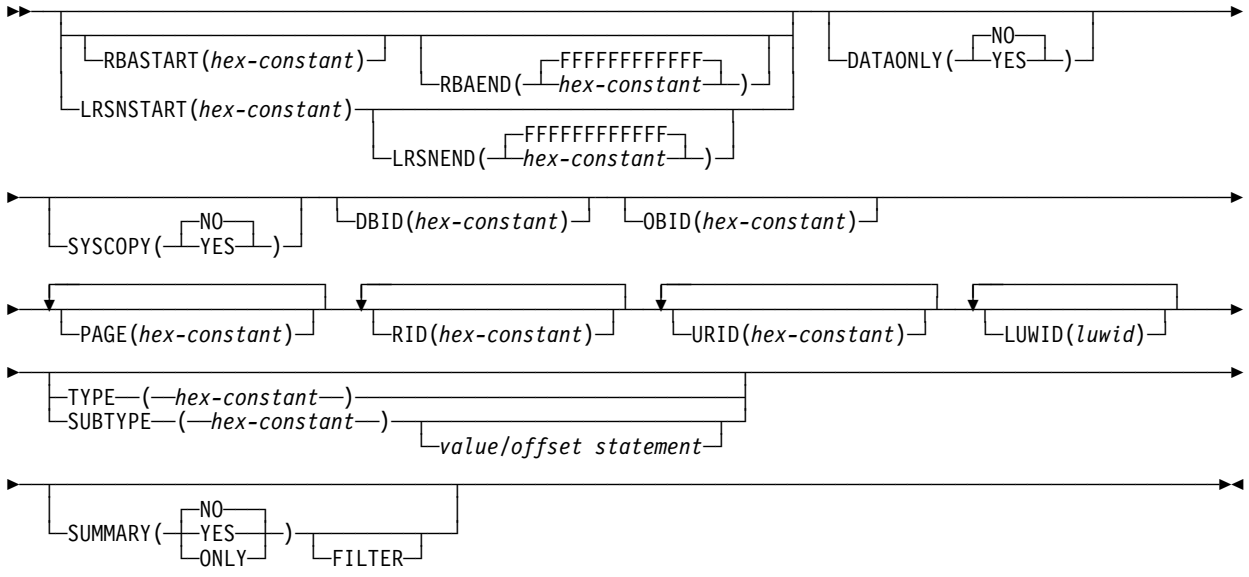
- One or more units of recovery identified by URID
- A single database

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.

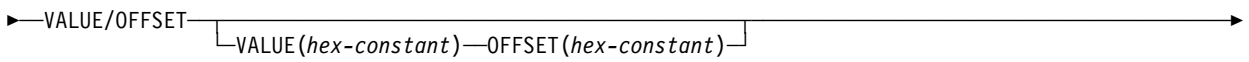
Syntax and options of the control statement

For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.

DSN1LOGP syntax diagram



value/offset statement:



Attention: If you specify more than one parameter:

- Separate them by commas (no blanks)
- Specify them in any order.

Option descriptions

To execute DSN1LOGP, construct a batch job. The utility name, DSN1LOGP, should appear on the EXEC statement, as shown in “Sample control statements” on page 481.

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.

The keywords are described below; alternative spellings or abbreviations are noted.

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), and 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), and 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 containing an LRSN value greater than or equal to the LRSN value specified on LRSNSTART. The default LRSN is the LRSN at the beginning of the data sets. 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 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 locate the log data sets.

LRSNEND(*hex-constant***)**

Specifies the LRSN value of the last log record to be scanned. When LRSNSTART is specified, the default is X'FFFFFFFFFFFF'. Otherwise, it is the end of the data sets. 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 records associated with that DBID. For any given job, specify this keyword only once.

hex-constant is a hexadecimal value consisting of 1 to 4 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 following SQL HEX function in a SELECT statement to convert a DBID to hexadecimal format:

```
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 records associated with that identifier.

hex-constant is a hexadecimal value consisting of 1 to 4 characters. Leading zeros are not required.

Whenever DB2 makes a change to data, the log record describing 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 you must include when you execute DSN1LOGP.

Whenever DB2 makes a change to an index, the log record describing 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 will 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 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, as well as the 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 cannot both be specified.

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 extracted to those associated with that particular record. The log records extracted include not only those directly associated with the RID, such as insert and delete, but also the control records 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. 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 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 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 hex characters. The last element of the LUWID is the commit sequence number of 4 hex characters, preceded by a period.

TYPE(*hex-constant***)**

Limits the log records 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	Allocating or deallocating 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 written for RMID 14
17	Checkpoint record of modified page set
19	Type 2 Index update
1A	Type 2 Index under/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

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 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 specified in the VALUE option must appear.

hex-constant is a hexadecimal value consisting of a maximum of 8 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.

Before running DSN1LOGP

This section contains information you need to know before you run DSN1LOGP.

Environment

DSN1LOGP runs as a batch MVS job.

You can use DSN1LOGP on archive data sets, but not active data sets, when DB2 is running.

Authorization required

DSN1LOGP requires no special authorization. However, if any of the data sets involved is RACF-protected, the authorization ID of the job must have RACF authority.

Control statement

See “Syntax and options of the control statement” on page 471 for DSN1LOGP syntax and option descriptions.

Required data sets

When you execute DSN1LOGP, you must provide the following DD statements:

SYSPRINT All error messages, exception conditions, and the detail report are written to the SYSPRINT file. The logical record length (LRECL) is 131.

SYSIN Keywords are specified in this file. The control statement keywords are described under “Option descriptions” on page 472. The LRECL must be 80. Keywords and values must appear in characters 1 through 72. You can specify as many as 50 control statements for a given job. All records are concatenated into a single string.

SYSSUMRY The formatted output of a summary report is written to the SYSSUMRY file. The LRECL is 131. For an example of the appropriate JCL, see page 482.

The recovery log is identified by DD statements described by 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 to process 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 also provide the beginning and ending relative byte addresses (RBAs) for the range of the recovery log you want displayed. DSN1LOGP then associates the beginning and ending RBA specifications you provide with the appropriate data set names.

See Example 1 on page 481 for guidance in using this DD statement.

ACTIVE*n* If the BSDS is not available, and if the active log data sets involved have been copied and sent to you, you can specify the set of active log data sets to be processed by DSN1LOGP by specifying one or more ACTIVE DD statements. If the REPRO command of access

method services was used to copy the active log to tape, 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; 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 the job indicates, but they must be in the correct log RBA sequence.

See Example 2 on page 481 for guidance in using these DD statements.

ARCHIVE If the BSDS is not available (as described under ACTIVE*n*, above), 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 as shown in Example 3 on page 481.

Each DD statement you include identifies another archive log data set. If you identify more than one archive log data set, you must list the DD statements corresponding 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 the job indicates, but they must be in the correct log RBA sequence.

See Example 3 on page 481 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

This section describes the following tasks associated with running the DSN1LOGP utility:

Reading archive log data sets on tape

If your archive logs are stored on tape, two files are constructed on tape during the archiving process. The first file is the BSDS, and the second is a dump of the active log that is currently being archived. If a failure occurs during the time the BSDS is being archived, DB2 might omit the BSDS. In this case, the first file contains the active log.

If archiving is performed on tape, the first letter of the lowest-level qualifier of the archived information varies for the first and second data sets on the tape. 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 data set names all end in A*xxxxxxx*, 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 MVS JES statements:

JES2 environment JCL	**Description**
/***SETUP**	Alert the MVS 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. This JCL is placed on the JOB card.
JES3 Environment JCL	**Description**
//***MAIN SETUP=JOB**	Alert the MVS 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. This JCL is placed on the JOB card.

Alternatively, you can submit the job to an MVS initiator that has been established by your operations center for exclusive use by jobs that require tape mounts. You can specify the initiator class using the CLASS parameter on the JOB card, in both JES2 and JES3 environments.

For additional information on these options, refer to the *OS/390 MVS JCL User's Guide* or the *OS/390 MVS JCL Reference.*

Locating table and index identifiers

You can 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 3-9. DSN1PRNT” on page 493.

Sample control statements

Example 1: Using DSN1LOGP with an available BSDS. This example shows how to extract the 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', for the table space or index space identified 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. Whenever 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 on page 429and see Section 4 (Volume 1) of *DB2 Administration Guide*. During normal DB2 operation, messages are issued that include information about log RBAs.

Example 2: Using DSN1LOGP on the active log (no BSDS available). This 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 identified by the DBID of X'10A' and the OBID of X'1F'. The only information that is extracted is information relating to page numbers X'3B' and X'8C'. 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: Using DSN1LOGP on archive log data (no BSDS available). This example 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)
/*
```

Example 4: Using 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.

This example produces both a detail and a summary report using 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 will be 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: Data sharing— using DSN1LOGP on all members of a data sharing group. This example shows extract log information pertaining to the table space 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: Data sharing— using DSN1LOGP on a single member of a data sharing group. This example shows extract log information pertaining to the table space 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

Interpreting 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.

Reviewing DSN1LOGP output

With the SUMMARY option, you can produce a summary report, a detail report, or both.

Figure 32 on page 485 shows a sample of the summary report. Figure 33 on page 486 shows a sample of the detail report. Figure 34 on page 491 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 on page Figure 32 on page 485 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 message:

  ```
  DSN1150I SUMMARY OF COMPLETED EVENTS
  ```

- The second section is headed by the message:

  ```
  DSN1157I RESTART SUMMARY
  ```

The first section lists all completed units of recovery (URs) and checkpoints within the range of the log scanned. Events are listed chronologically, with URs listed by the order of their completion and checkpoints listed when the end of a checkpoint is processed. The page sets changed by each completed UR are listed. If a log record associated with a UR is unavailable (as it would be if, for example, the range of the log scanned is not large enough to contain all records for a given UR), the attribute INFO=PARTIAL is displayed for the UR. Otherwise, the UR is marked INFO=COMPLETE.

The DISP attribute can be one of the following: COMMITTED, ABORTED, INFLIGHT, IN-COMMIT, IN-ABORT, or POSTPONED ABORT. 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 required of DB2 at restart as it is recorded in the log you specified. If the log is available, the checkpoint 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 required to complete those writes. If a log record 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=V61B     UR  CONNID=V61B      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=V61B     UR  CONNID=V61B      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=V61B
               STARTRBA=000000068CD3  ENDRBA=00000006CAED  STARTLRSN=AA527AA809DF  ENDLRSN=AA527AA829F4
               DATE=94.347  TIME=12:32:29

DSN1162I DSN1LPRT  MEMBER=V61C     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=V61A     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 32. Sample DSN1LOGP summary report

Description of the detail report

The *detail report* on page Figure 33 includes the following records:

- *Redo/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 an optional parameter.

```
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 33 (Part 1 of 5). Sample DSN1LOGP detail report

```
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

......
0000000110A0  MEMBER(M02     )  URID(0000000110A0)  LRSN(AB625379B94A)
     TYPE(UR CONTROL)  SUBTYPE(BEGIN UR)
     *LRH* 009000A0 00200001 03800000 000110A0 00000000 00000126 00000000 0000AB62
           5379B94A 0002
     0000  00020000 00004000 00000000 00000700 0000F0F2 F14BD6D7 D5D3C7D9 F0F0E2E8
     0020  E2D6D7D9 4040AB62 5379B945 07044040 40404040 40400000 00000000 0000E5F4
     0040  F2C44040 40400000 00000000 0000001A 0001E4E2 C9C2D4E2 E840E2E8 C5C3F1C4
     0060  4040AB62 53782F9A 0001

000000011130  MEMBER(M02     )  URID(0000000110A0)  LRSN(AB625379B955)  DBID(0001)  OBID(00CF)  PAGE(00000009)
     TYPE( UNDO  REDO )  SUBTYPE(INSERT IN A DATA PAGE)  CLR(NO)  PROCNAME(DSNISMRT)
     *LRH* 00740090 06000001 0E800000 000110A0 00000001 10A00126 00000001 10A0AB62
           5379B955 0002
     *LG** 90000100 CF000009 3100AB62 51C6F5F1 0000
     0000  003C5036 00D10000 00003400 D1360113 0005F0F7 F1F8F9F5 F0F1F4F5 F2F8F4F8
     0020  00000001 0EB70000 00000000 8000AB62 53782F89 00000000 00000002

0000000111A4  MEMBER(M02     )  LRSN(AB625379D424)  DBID(0001)  OBID(0087)
     TYPE(PAGE SET CONTROL)  SUBTYPE(PAGE SET OPEN)
     *LRH* 00A00074 00020001 0E800000 00000000 00000001 10000126 00000001 1000AB62
           5379D424 0002
     0000  00010087 6C010100 000000CF 0040C4E2 D5C4C2F0 F140C4E2 D5D3D3E7 F0F10001
     0020  00D10000 10009200 00130020 00000000 00000000 00000000 00000000 00000000
     0040  00000000 00000010 00010000 00000000 00000000 00000000 00000000 00000000
     0060  00AA6F9F 3E74EFAA 80DE586F F5000000 0000C4E2 D5C3F4F1 F040

000000011244  MEMBER(M02     )  URID(0000000110A0)  LRSN(AB625379D47D)  DBID(0001)  OBID(0087)  PAGE(00000005)
     TYPE( UNDO  REDO )  SUBTYPE(TYPE 2 INDEX UPDATE)  CLR(NO)  PROCNAME(DSNKINSL)
     *LRH* 005700A0 06000019 0E800000 000110A0 00000001 11300126 00000001 1130AB62
           5379D47D 0002
     *LG** 00000100 87000005 64000000 00000000 0000
     0000  001F1000 007F0001 00000936 000E2000 00011300 05800000 02549DAC 87D076

00000033592A0  MEMBER(M01     )  LRSN(AB62537A111E)  DBID(0001)  OBID(0086)
     TYPE(PAGE SET CONTROL)  SUBTYPE(PAGE SET STATUS RECORD)
     *LRH* 00660066 00020009 0E800000 00000000 00000335 923A0126 00000335 923AAB62
           537A111E 0001
     0000  00010086 C4E2D5C4 C2F0F140 C4E2D5D3 D3E7F0F2 00010000 92018000 00000334
     0020  EC3AAB62 5260AB0B 00000000 00000000 00000000 00000000 00000000 00000000

000000001129B  MEMBER(M02     )  LRSN(AB62537B40E6)  DBID(0001)  OBID(0086)
     TYPE(PAGE SET CONTROL)  SUBTYPE(PAGE SET OPEN)
     *LRH* 00A00057 00020001 0E800000 00000000 00000001 11A40126 00000001 11A4AB62
           537B40E6 0002
     0000  00010086 6C010100 000000CF 0040C4E2 D5C4C2F0 F140C4E2 D5D3D3E7 F0F20001
     0020  00D10000 10009200 00130020 00000000 00000000 00000000 00000000 00000000
     0040  00000000 00000010 00010000 00000000 00000000 00000000 00000000 00000000
     0060  00AB624E 7933BEAB 62501309 25000000 0000C4E2 D5C3F4F1 F040

000000001133B  MEMBER(M02     )  URID(0000000110A0)  LRSN(AB62537B4242)  DBID(0001)  OBID(0086)  PAGE(00000005)
     TYPE( UNDO  REDO )  SUBTYPE(TYPE 2 INDEX UPDATE)  CLR(NO)  PROCNAME(DSNKINSL)
     *LRH* 005300A0 06000019 0E800000 000110A0 00000001 12440126 00000001 1244AB62
           537B4142 0002
     *LG** 00000100 86000005 64000000 00000000 0000
     0000  001B1000 007F0001 00000936 000A2000 00011300 05AB6253 782F89
```

Figure 33 (Part 2 of 5). Sample DSN1LOGP detail report

```
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

000000003956D  MEMBER(M02     )  LRSN(AB6254EE48E9)  DBID(0006)  OBID(0009)
     TYPE( CHECKPOINT)  SUBTYPE(DBE TABLE WITH EXCEPTION DATA)
     *LRH* 0061003E 2100001D 0E800000 00000000 00000003 952F0126 00000003 952FAB62
           54EE48E9 0002
     0000  00000000 C4E2D5C4 C2F0F640 E2E8E2C4 C2C1E2C5 00060009 00000000 00000000
     0020  00000000 00000000 00000000 00000000 00000000 00000000 000000

0000000395CE  MEMBER(M02     )  LRSN(AB6254EE48F2)  DBID(0006)  OBID(0009)
     TYPE( CHECKPOINT)  SUBTYPE(DBE TABLE WITH PIECE DATA)
     *LRH* 01F60061 2100001E 0E800000 00000000 00000003 956D0126 00000003 956DAB62
           54EE48F2 0002
     0000  00000600 09C4E2D5 C4C2F0F6 40E2E8E2 C4C2C1E2 C5000000 0020FFFF FFFFFFFF
     0020  00000000 00000000 000001BC 000001D4 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

 .....
```

Figure 33 (Part 3 of 5). Sample DSN1LOGP detail report

```
000000057B32  MEMBER(M02     )  LRSN(AB62564FD672)  DBID(0001)  OBID(00CF)
     TYPE(PAGE SET CONTROL)  SUBTYPE(PAGE SET OPEN)
     *LRH* 00A0006C 00020001 0E800000 00000000 00000005 7AC60126 00000005 7AC6AB62
           564FD672 0002
     0000  000100CF 6C010100 00000000 0040C4E2 D5C4C2F0 F140E2E8 E2D3C7D9 D5E70001
     0020  00000000 10008000 00130020 00000000 00000000 00000000 2A0C0000 00000000
     0040  00000000 00000010 00010000 00000000 00000000 00000000 00000000 00000000
     0060  00AB6251 91E1F9AB 62560CB6 B8000000 0000C4E2 D5C3F4F1 F040

000000057BD2  MEMBER(M02     )  URID(000000057BD2)  LRSN(AB62564FD6D3)
     TYPE(UR CONTROL)  SUBTYPE(BEGIN UR)
     *LRH* 007400A0 00200001 03800000 00057BD2 00000000 00000126 00000000 0000AB62
           564FD6D3 0002
     0000  00070000 00004000 00000000 00000700 0000F0F2 F14BC3D3 E2D3C7D9 F0F0E2E8
     0020  E2D6D7D9 4040AB62 564FD6D0 F7034040 40404040 40400000 00000000 0000E5F4
     0040  F2C44040 40400000 00000000 0000

000000057C46  MEMBER(M02     )  URID(000000057BD2)  LRSN(AB62564FD6DD)  DBID(0001)  OBID(00CF)  PAGE(00000009)
     TYPE( UNDO  REDO )  SUBTYPE(UPDATE NOT IN-PLACE , DATA PART ONLY IN A DATA PAGE)  CLR(NO)  PROCNAME(DSNIREPR)
     *LRH* 006C0074 06000001 0E800000 00057BD2 00000005 7BD20126 00000005 7BD2AB62
           564FD6DD 0002
     *LG** 90000100 CF000009 2D00AB62 54DA2429 0000
     0000  00346137 00D18200 00210013 0013057A 9C8001AB 625391D6 C4AB6256 4FB04700
     0020  02000000 8001AB62 5391D6C4 00000000 00000002

000000057CB2  MEMBER(M02     )  URID(000000057BD2)  LRSN(AB62564FD6FA)
     TYPE(UR CONTROL)  SUBTYPE(BEGIN COMMIT1)
     *LRH* 005C006C 00200002 03800000 00057BD2 00000005 7C460126 00000005 7C46AB62
           564FD6FA 0002
     0000  00070000 00004000 00000000 00000700 0000F0F2 F14BC3D3 E2D3C7D9 F0F04040
     0020  40404040 40400000 00000000 00000000 00000000 0000

000000057D0E  MEMBER(M02     )  URID(000000057BD2)  LRSN(AB62564FD709)
     TYPE(UR CONTROL)  SUBTYPE(PHASE 1 TO 2)
     *LRH* 0034005C 0020000C 03800000 00057BD2 00000005 7CB20126 00000005 7CB2AB62
           564FD709 0002
     0000  00070000 00004000 00000000 0000

000000057D42  MEMBER(M02     )  LRSN(AB62564FE492)  DBID(0001)  OBID(00CF)
     TYPE(PAGE SET CONTROL)  SUBTYPE(PAGE SET WRITE)
     *LRH* 006C0034 00020007 0E800000 00000000 00000005 7B320126 00000005 7B32AB62
           564FE492 0002
     0000  0000F5E2 C3D40001 00CFC4E2 D5C4C2F0 F140E2E8 E2D3C7D9 D5E70000 00000000
     0020  0000AB62 564FD672 AB62564F D6720000 00000000 01017EFD EAB80000 09000009
     0040  AB62564F D6DD

000000057DAE  MEMBER(M02     )  URID(000000057BD2)  LRSN(AB62564FE4C9)
     TYPE(UR CONTROL)  SUBTYPE(END COMMIT2)
     *LRH* 0034006C 00200010 03800000 00057BD2 00000005 7D0E0126 00000005 7D0EAB62
           564FE4C9 0002
     0000  00070000 00004000 00000000 0000

000000057DE2  MEMBER(M02     )  LRSN(AB62564FE550)  DBID(0115)  OBID(0002)
     TYPE(PAGE SET CONTROL)  SUBTYPE(PAGE SET CLOSE)
     *LRH* 002A0034 00020003 0E800000 00000000 00000005 7D420126 00000005 7D42AB62
           564FE550 0002
     0000  01150002

000000057E0C  MEMBER(M02     )  LRSN(AB62564FEAB1)  DBID(0113)  OBID(000C)
     TYPE(PAGE SET CONTROL)  SUBTYPE(PAGE SET WRITE)
     *LRH* 006C002A 00020007 0E800000 00000000 00000005 7DE20126 00000005 7DE2AB62
           564FEAB1 0002
     0000  0000F5D7 C3D60113 000CC4C2 C6E6F0F0 F1F1C9E4 C6E6F0F0 F0F30000 00000000
     0020  00000000 0004D98E 00000004 D98E0000 00000000 11010000 00000000 03000003
     0040  AB62553F 9821
```

Figure 33 (Part 4 of 5). Sample DSN1LOGP detail report

```
000000057E78  MEMBER(M02     )  LRSN(AB62564FF072)  DBID(0113)  OBID(000C)
     TYPE(PAGE SET CONTROL)  SUBTYPE(PAGE SET CLOSE)
     *LRH* 002A006C 00020003 0E800000 00000000 00000005 7E0C0126 00000005 7E0CAB62
           564FF072 0002
     0000  0113000C

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 33 (Part 5 of 5). Sample DSN1LOGP detail report

Interpreting data propagation information in the summary report

The sample output on page Figure 34 on page 491 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:

- START RBA and END RBA show the first and last RBAs captured for the unit of recovery that was not retrieved. The range that the start and end RBA cover can include one or all of the SQL statements within the scope of the unit of recovery.
- TABLE LIST OVERFLOW tells whether more than 10 distinct data capture table IDs were updated by this unit of recovery. This example shows no overflow occurred.
- LR WRITTEN shows the number of written log records that represented changes to tables defined for data capture and were available to the DB2CDCEX routine. Recursive SQL changes from DB2CDCEX and changes from other attachments 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). This example 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 34. Sample data propagation information from the summary report

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 it issues, see Appendix C (Volume 2) of *DB2 Administration Guide*.

DSN1LOGP can abend with a user abend code of X'099'. You can find the corresponding abend reason code in register 15 (at the time of error).

Chapter 3-9. 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

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.

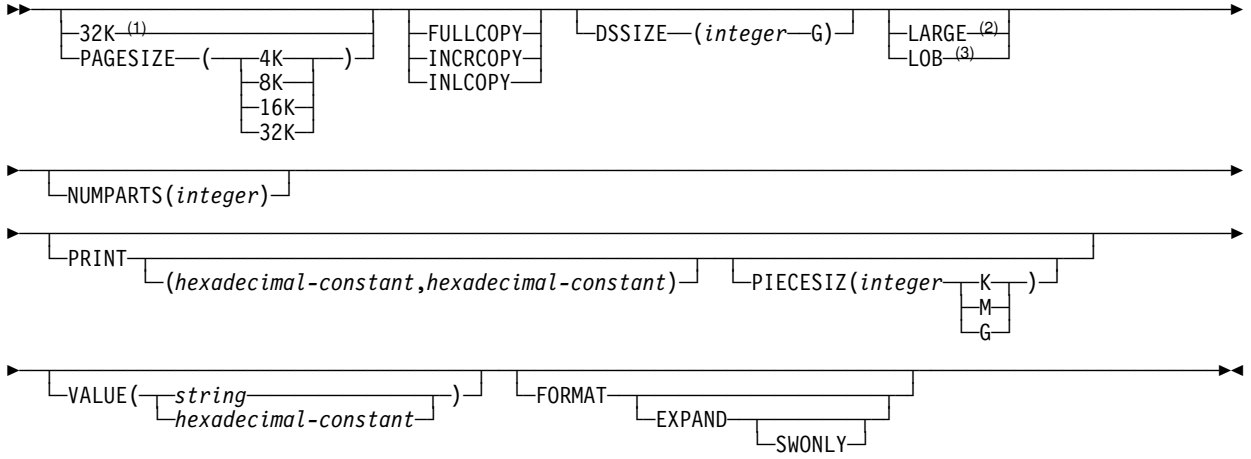
DSN1PRNT is especially useful when you want to identify the contents of a table space or index. You can run DSN1PRNT on image copy data sets as well as table spaces and indexes. DSN1PRNT accepts an index image copy as input when you specify the FULLCOPY option.

DSN1PRNT is compatible with LOB table spaces, when you specify the LOB keyword, and omit the INLCOPY keyword.

Syntax and options of the control statement

For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.

DSN1PRNT syntax diagram



Notes:

1 The preferred option is PAGESIZE(32K).

2 The preferred method of specifying LARGE is DSSIZE(4G).

3 Do not specify INLCOPY with the LOB parameter.

Important: If you specify more than one parameter:

- Separate them by commas (no blanks).
- Specify them in any order.

Option descriptions

Specify one or more of the parameters listed below on the EXEC card to run DSN1PRNT.

32K Specifies that the SYSUT1 data set has a 32-KB page size. If the SYSUT1 data set has a 32-KB page size and you do not specify this option, DSN1PRNT may produce unpredictable results, because the default page size is 4 KB.

The preferred option is **PAGESIZE(32K)**.

PAGESIZE Specifies the page size of the input data set that is defined by SYSUT1. If you specify an incorrect page size, DSN1PRNT may produce unpredictable results.

If you omit PAGESIZE, DSN1PRNT tries to determine the page size from the input data set. 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.

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 unless the input data set is a LOB, in which case DB2 assumes a 4 GB input data set size.

integer must match the DSSIZE value 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.

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, then DB2 assumes that the data set has a 4-GB boundary.

The preferred 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.

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.

NUMPARTS(*integer***)**

Specifies the number of partitions associated with the input data set. NUMPARTS is **required** if the input data set is partitioned. Valid specifications range from 1 to 254. DSN1PRNT uses this value to help locate the first page in a range to be printed. If you omit NUMPARTS or specify it as 0, DSN1PRNT will assume 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 indicating 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. 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 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)
```

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, REPLACE, and DELETE operations” on page 359.

PIECESIZ(*integer***)**

Specifies the maximum piece size (data set size) for non-partitioned indexes. The value you specify must match the value specified when the nonpartitioning index was created or altered.

The defaults for PIECESIZ are 2G (2 GB) for indexes backed by non-large table spaces and 4G (4 GB) for indexes 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 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 input data set SYSUT1 to be scanned for the character string 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 characters.

(*hexadecimal-constant***)**
Can consist of 2 to 40 hexadecimal characters. You must specify two single quotation mark characters before and after the hexadecimal character string.

If, for example, you want to search your input file for the string '12345', your JCL might look like the following example:

```
//STEP1 EXEC PGM=DSN1PRNT,PARM='VALUE(12345)'
```

On the other hand, you might want to search for the equivalent hexadecimal character string, in which case your JCL might look like this:

```
# //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. This option is intended to be used only under the direction of your IBM Support Center.

SWONLY
Causes DSN1PRNT to use software to expand the compressed data, even when the compression hardware is available. This option is intended to be used only under the direction of your IBM Support Center.

FULLCOPY Specifies that a DB2 full image copy (not a DFSMS concurrent copy) of your data is 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 using 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 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 using 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 an inline copy data set.

When you use DSN1PRNT to print a page or a page range from an inline copy produced by LOAD or REORG, DSN1PRNT prints all instances of the pages. The last instance of the page or pages
| printed is the last one created by the utility.

Before running DSN1PRNT

This section contains information you need to know before you run DSN1PRNT.

Environment

Run DSN1PRNT as an MVS 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 you want to print.

Authorization required

None is required. However, if any of the data sets is RACF-protected, the authorization ID of the job must have RACF authority.

Control statement

See “Syntax and options of the control statement” on page 493 for DSN1PRNT syntax and option descriptions.

Required data sets: DSN1PRNT uses the DD cards described below:

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. DSN1PRNT assumes that block size is a multiple of 4096 bytes (as is standard for DB2 data sets).

Disposition for this data set must be specified as OLD (DISP=OLD) to ensure that it is not in use by DB2. Disposition for this data set must be specified as SHR (DISP=SHR) only in circumstances where the DB2 STOP DATABASE command does not work.

The requested operation takes place only for the data set specified. If the input data set belongs to a linear table space or index space that is larger than 2 gigabytes, 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. (In other words, `DSN=...A002`.)

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. This is because DSN1COPY scans the whole SYSUT1 data set, but DSN1PRNT may be able to stop scanning before the end. 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 following query on the DB2 catalog to get the information you need:

```
SELECT I.CREATOR,
      I.NAME,
      S.PGSIZE,
      CASE S.DSSIZE
        WHEN 0 THEN CASE S.TYPE
                      WHEN ' ' THEN 2097152
                      WHEN 'I' THEN 2097152
                      WHEN 'L' THEN 4194304
                      WHEN 'K' THEN 4194304
                      ELSE NULL
                    END
        ELSE S.DSSIZE
      END
 FROM SYSIBM.SYSINDEXES I,
      SYSIBM.SYSTABLES  T,
      SYSIBM.SYSTABLESPACE S
 WHERE I.CREATOR='DSN8610' AND
       I.NAME='XEMP1' AND
       I.TBCREATOR=T.CREATOR AND
       I.TBNAME=T.NAME AND
       T.DBNAME=S.DBNAME AND
       T.TSNAME=S.NAME;
```

Sample control statements

Example 1: Running DSN1PRNT

```
//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.SYSUTIL.I0001.A001,DISP=SHR
```

Example 2: Printing a nonpartitioning index with a 64 MB piece size

```
//PRINT2 EXEC PGM=DSN1PRNT,
//            PARM=(PRINT(F0000,F000F),FORMAT,PIECESIZ(64M))
//* PRINT 16 PAGES IN THE 61ST PIECE OF AN NPI WITH PIECE SIZE OF 64M
//SYSUDUMP DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DISP=OLD,DSN=DSNCAT.DSNDBD.MMRDB.NPI1.I0001.A061
```

DSN1PRNT output

Interpreting 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.

Chapter 3-10. DSN1SDMP

Under the direction of the IBM Support Center, use the IFC Selective Dump (DSN1SDMP) utility to:

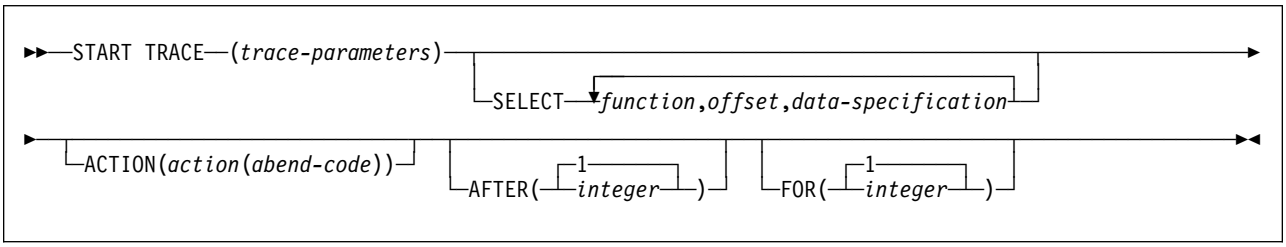
- Force dumps when selected DB2 trace events occur.
- Write DB2 trace records to a user-defined MVS data set.

For information about the format of trace records, see Appendix D (Volume 2) of *DB2 Administration Guide*.

Syntax and options of the control statement

For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.

DSN1SDMP syntax diagram



Option descriptions

START TRACE (*trace-parameters***)**

START TRACE is a required keyword and must be the first keyword specified in the SDMPIN input stream. The trace parameters that you use are those described in Chapter 2 of *DB2 Command Reference*, except you do not use the subsystem recognition character.

If the START TRACE command in the SDMPIN input stream is not valid, or 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 buffer (OP). OP buffer destinations are specified in the DEST keyword of -START TRACE. There are eight OP buffer destinations, 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 specified on the START TRACE command. SELECT expands the data available to select on in a trace record and allows more specific selection of data in the trace record than using START TRACE alone. A maximum of eight SELECT criteria can be specified.

The selection criteria use the concept of the current-record pointer. The current-record pointer is initialized to zero, meaning the beginning of the trace record. For this instance of the DSN1SDMP trace, the trace record begins with the self-defining section. For information on the fields in the DB2 trace records, see Appendix D (Volume 2) of *DB2 Administration Guide*.

The selection criteria are specified with the following parameters:

function Specifies the type of search to be performed on the trace record. The specified value must be two characters. The possible values are:

DR Direct comparison of data from the specified offset. The offset is always calculated from the current-record pointer.

The current-record pointer locates the point in the trace record where the offset is calculated. The current-record pointer is initialized to zero (the start of the trace record) and is modified by previous P*x* and LN functions (below).

GE Greater than or equal comparison of data from 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 is specified on the SELECT option.

The current-record pointer locates the point in the trace record where the offset is calculated. The current-record pointer is initialized to zero (the start of the trace record) and is modified by previous P*x* and LN functions (below).

LE Less than or equal comparison of data from 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 is specified on the SELECT option.

The current-record pointer locates the point in the trace record where the offset is calculated. The current-record pointer is initialized to zero (the start of the trace record) and is modified by previous P*x* and LN functions (below).

P1, P2, or P4

Selects the 1, 2, or 4 byte field located *offset* bytes past the start of the record. 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 you specify).

This offset is saved as the current-record pointer to be used on subsequent DR, LE, GR, and LN requests.

For example, suppose the user knows the offset to the standard header is 4 bytes long and is located in the first four 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 indicated in the 2-byte field located *offset* bytes from the previous current-record pointer.

This offset is saved as the current-record pointer to be used on subsequent DR, LE, GR, and LN requests.

offset A decimal value that specifies the number 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:

Figure 35. Format of the DB2 trace record

- The format of the self-defining section depends on the trace type.
- The format and contents of the data sections depend on the IFCID 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 on 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(*action***(***abend-code***))**

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, and it should be used with extreme caution. Not all abends are recoverable, even if the ABENDRET parameter is specified. Some abends may 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*)
Possible values for *action* are:

ABENDRET ABEND and retry the agent.

ABENDTER ABEND and terminate the agent.

If *action* is not specified, the record is written with no action performed.

An abend reason code can also be specified on this parameter. The codes must be in the range `00E60100-00E60199`. If no abend code is specified, `00E60100` is used.

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***)**
FOR is an optional keyword that specifies the number of times that the ACTION is to take place on reaching the specified trace point. 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**).

Before running DSN1SDMP

This section contains information you need to know before you run DSN1SDMP.

Environment

Run DSN1SDMP as an MVS job and execute it with the DSN TSO command processor. To execute DSN1SDMP, the DB2 subsystem must be running.

The MVS job completes only under the following conditions:

- The TRACE and any additional selection criteria started by DSN1SDMP meet the criteria specified in the FOR parameter.
- The TRACE started by DSN1SDMP is stopped 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 specified in the *trace-parameters* of the START TRACE keyword.

Control statement

See "Syntax and options of the control statement" on page 501 for DSN1SDMP syntax and option descriptions.

Required data sets: DSN1SDMP uses the DD cards described below:

SDMPIN Defines the control data set that specifies the input parameters to DSN1SDMP. This DD card is required. The LRECL is 80. Only the first 72 columns are checked by DSN1SDMP.

SDMPPRNT Defines the sequential message data set used for DSN1SDMP messages. If the SDMPPRNT DD statement is omitted, no messages are written. The LRECL is 131.

SYSABEND Defines the data set to contain an ABEND dump in case DSN1SDMP abends. This DD card is optional.

SDMPTRAC Defines the sequential DB2 trace record data set used for trace records returned to DSN1SDMP from DB2. This DD card 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 written to SDMPTRAC are of the same format as records written to SMF or GTF, except that, instead of containing the SMF or GTF headers, the SDMPTRAC trace records contain the monitor header (mapped by DSNDQWIW). The DCB parameters are VB, BLKSIZE=8192, LRECL=8188.

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 only used to connect to DB2.

When no plan name is specified on the DSN RUN command, DSN defaults the plan name to the program. 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

This section describes the following tasks associated with running the DSN1SDMP utility:

"Assigning buffers" on page 506
"Generating a dump" on page 506
"Stopping or modifying DSN1SDMP traces" on page 506

Assigning buffers

The OPX trace destination assigns the next available OP buffer. You must specify the OPX destination for all traces 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. MONITOR-type traces default to 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 page 501). 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. The default buffer size is 8KB. 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, you will receive error message DSN2724I.

Generating a dump

All of the following must occur before DSN1SDMP generates a DB2 dump:

- DB2 produces a trace record that satisfies all of the selection criteria.
- An abend action (ABENDRET or ABENDTER) is specified.
- The AFTER and FOR conditions for the trace are satisfied.

If all of these three things 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; for example:

```
-STOP TRACE=P CLASS(32)
```

DSN1SDMP is designed to execute 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 started
by DSN1SDMP.

A STOP TRACE or MODIFY TRACE command entered from a console against the
trace 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 associated with the DSN1SDMP STOP TRACE command will vary depending on the specific trace command entered by the console operator.

If the console operator terminates the original trace using the STOP TRACE command, the subsequent STOP TRACE command issued by DSN1SDMP fails.

If the console operator enters a MODIFY TRACE command, the modified trace might also be terminated by the STOP TRACE command issued by DSN1SDMP if MODIFY TRACE processing completes before the DSN1SDMP command is issued.

Sample control statements

Example 1: Skeleton JCL for DSN1SDMP

```
//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=8188,RECFM=VB,BLKSIZE=8192)
//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
//*
```

Example 2: SDMPIN for ABEND and TERMINATE AGENT on -904 SQL CODE

```
//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'
/*
```

Example 3: SDMPIN for ABEND and RETRY on RMID 20

```
//*       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'
/*
```

Example 4: Dump on SQLCODE -811 RMID16 IFCID 58

```
//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))
/*
```

DSN1SDMP output

Interpreting 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.

Appendixes

Appendix A. Limits in DB2 for OS/390

System storage limits might preclude the limits specified here. The limit for items not specified below is system storage.

Table 78. Identifier length limits

Item	Limit
Longest collection ID, correlation name, statement name, or name of an alias, column, cursor, index, table, table check constraint, stored procedure, synonym, user-defined function	18 bytes
Longest authorization name, or name of a database, package, plan, referential constraint, schema, storage group, or trigger	8 bytes
Longest host identifier	64 bytes
Longest server name or location identifier	16 bytes

Table 79. 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}
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 80 (Page 1 of 2). String length limits

Item	Limit
Maximum length of CHAR	255 bytes
Maximum length of GRAPHIC	127 DBCS characters
Maximum length of VARCHAR[5]	4046 bytes for 4-KB pages 8128 bytes for 8-KB pages 16320 bytes for 16-KB pages 32704 bytes for 32-KB pages

Table 80 (Page 2 of 2). String length limits

Item	Limit
Maximum length of VARGRAPHIC[5]	4046 bytes (2023 DBCS characters) for 4-KB pages 8128 bytes (4064 DBCS characters) for 8-KB pages 16320 bytes (8160 DBCS characters for 16-KB pages 32704 bytes (16352 DBCS characters) for 32-KB pages
Maximum length of CLOB	2 147 483 647 bytes (2 gigabytes - 1 byte)
Maximum length of DBCLOB	1 073 741 824 DBCS characters
Maximum length of BLOB	2 147 483 647 bytes (2 gigabytes - 1 byte)
Maximum length of a character constant	255 bytes
Maximum length of a hexadecimal constant	254 digits
Maximum length of a graphic string constant	124 DBCS characters
Maximum length of a concatenated character string	2 147 483 647 bytes (2 gigabytes - 1 byte)
Maximum length of a concatenated graphic string	1 073 741 824 DBCS characters
Maximum length of a concatenated binary string	2 147 483 647 bytes (2 gigabytes - 1 byte)

Table 81. Datetime limits

Item	Limit
Smallest DATE value (shown in ISO format)	0001-01-01
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 82 (Page 1 of 2). DB2 limits on SQL statements

Item	Limit
Maximum number of columns 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 6 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 2G 254 for table spaces that are defined with LARGE or a DSSIZE greater than 2G

[5] 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 82 (Page 2 of 2). DB2 limits on SQL statements

Item	Limit
Maximum size of a partition (table space or index)	For table spaces that are not defined with LARGE or a DSSIZE greater than 2G: 4 gigabytes, for 1 to 16 partitions 2 gigabytes, for 17 to 32 partitions 1 gigabyte, for 33 to 64 partitions For table spaces that are defined with LARGE: 4 gigabytes, for 1 to 254 partitions For table spaces that are defined with a DSSIZE greater than 2G: 64 gigabytes, for 1 to 254 partitions
Maximum size of a DBRM entry	131072 bytes
Longest index key	255 bytes less the number of key columns that allow nulls.
Maximum number of bytes used in the partitioning of a partitioned index[6]	255
Maximum number of columns in an index key	64
Maximum number of tables in a FROM clause	15
Maximum number of subqueries in a statement	14
Maximum total length of host and indicator variables pointed to in an SQLDA	32767 bytes 2147483647 bytes (2 gigabytes - 1 byte) for a LOB, subject to the limitations imposed by the application environment and host language
Longest host variable used for insert or update	32704 bytes for a non-LOB 2147483647 bytes (2 gigabytes - 1 byte) for a LOB, subject to the limitations imposed by the application environment and host language
Longest SQL statement	32765 bytes
Maximum number of elements in a select list	750
Maximum number of predicates in a WHERE or HAVING clause	750
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 column function)	4000 bytes
Maximum length of a table 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 2147483647 bytes (2 gigabytes - 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	254 bytes

Table 83. 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 used by each DB2 or application agent
Largest table or table space	16 terabytes
Largest log space	2^{48}
Largest active log data set	2 gigabytes
Largest archive log data set	2 gigabytes
Maximum number of active log copies	2
Maximum number of archive log copies	2
Maximum number of active log data sets (each copy)	31
Maximum number of archive log volumes (each copy)	1000
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	65279
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 gigabytes
Maximum partitioned data set size	See item “maximum size of a partition” in Table 82 on page 514
Maximum LOB data set size	64 gigabytes

[6] If the key of a partitioned index is longer than 255 bytes, only the first 255 bytes are used to determine the high value for each partition.

Appendix B. 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. 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 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

DSNUTILS ***must*** run in a WLM environment.

Authorization required

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, the privilege set must also include the authorization to run the specified utility.

Control statement

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 84 on page 518 for information about the default data dispositions specified for dynamically allocated data sets.

Table 84. Data dispositions for dynamically allocated data sets

ddname	**CHECK DATA**	**CHECK INDEX or CHECK LOB**	**COPY**	**LOAD**	**MERGECOPY**	**REBUILD INDEX**	**REORG INDEX**	**REORG TABLESPACE**
SYSREC	ignored	ignored	ignored	OLD KEEP KEEP	ignored	ignored	ignored	NEW CATLG CATLG
SYSDISC	ignored	ignored	ignored	NEW CATLG CATLG	ignored	ignored	ignored	NEW CATLG CATLG
SYSPUNCH	ignored	ignored	ignored	ignored	ignored	ignored	ignored	NEW CATLG CATLG
SYSCOPY	ignored	ignored	NEW CATLG CATLG	NEW CATLG CATLG	NEW CATLG CATLG	ignored	ignored	NEW CATLG CATLG
SYSCOPY2	ignored	ignored	NEW CATLG CATLG	NEW CATLG CATLG	NEW CATLG CATLG	ignored	ignored	NEW CATLG CATLG
SYSRCPY1	ignored	ignored	NEW CATLG CATLG	NEW CATLG CATLG	NEW CATLG CATLG	ignored	ignored	NEW CATLG CATLG
SYSRCPY2	ignored	ignored	NEW CATLG CATLG	NEW CATLG CATLG	NEW CATLG CATLG	ignored	ignored	NEW CATLG CATLG
SYSUT1	NEW DELETE CATLG	NEW DELETE CATLG	ignored	NEW DELETE CATLG	ignored	NEW DELETE CATLG	NEW CATLG CATLG	NEW DELETE CATLG
# SORTOUT	NEW DELETE CATLG	ignored	ignored	NEW DELETE CATLG	ignored	ignored	ignored	NEW DELETE CATLG
SYSMAP	ignored	ignored	ignored	NEW CATLG CATLG	ignored	ignored	ignored	ignored
SYSERR	NEW CATLG CATLG	ignored	ignored	NEW CATLG CATLG	ignored	ignored	ignored	ignored
# FILTER # #	ignored	ignored	NEW DELETE CATLG	ignored	ignored	ignored	ignored	ignored

If the DSNUTILS stored procedure restarts a current utility, refer to Table 85 on page 519 for information about the default data dispositions specified for dynamically-allocated data sets.

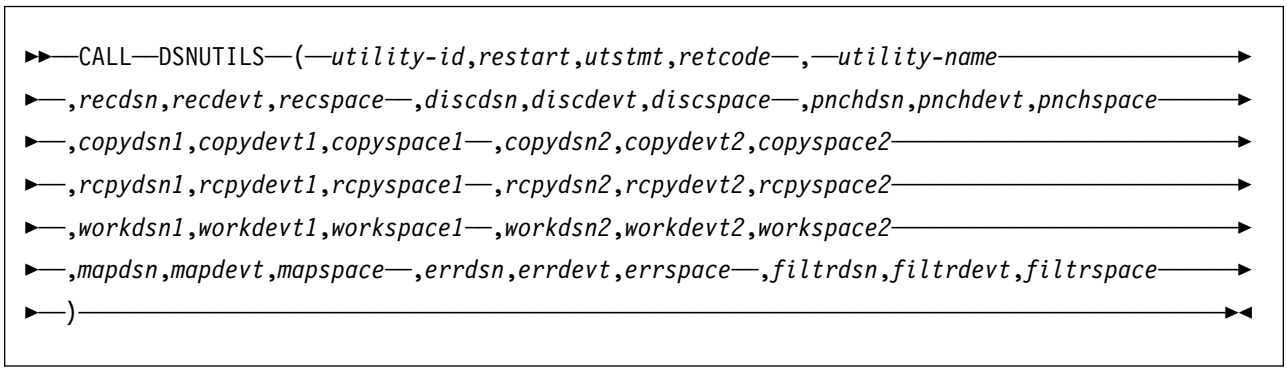
Table 85. Data dispositions for dynamically allocated data sets on RESTART

ddname	CHECK DATA	CHECK INDEX or CHECK LOB	COPY	LOAD	MERGECOPY	REBUILD INDEX	REORG INDEX	REORG TABLESPACE
SYSREC	ignored	ignored	ignored	OLD KEEP KEEP	ignored	ignored	ignored	MOD CATLG CATLG
SYSDISC	ignored	ignored	ignored	MOD CATLG CATLG	ignored	ignored	ignored	MOD CATLG CATLG
SYSPUNCH	ignored	ignored	ignored	ignored	ignored	ignored	ignored	MOD CATLG CATLG
SYSCOPY	ignored	ignored	MOD CATLG CATLG	MOD CATLG CATLG	MOD CATLG CATLG	ignored	ignored	MOD CATLG CATLG
SYSCOPY2	ignored	ignored	MOD CATLG CATLG	MOD CATLG CATLG	MOD CATLG CATLG	ignored	ignored	MOD CATLG CATLG
SYSRCPY1	ignored	ignored	MOD CATLG CATLG	MOD CATLG CATLG	MOD CATLG CATLG	ignored	ignored	MOD CATLG CATLG
SYSRCPY2	ignored	ignored	MOD CATLG CATLG	MOD CATLG CATLG	MOD CATLG CATLG	ignored	ignored	MOD CATLG CATLG
SYSUT1	MOD DELETE CATLG	MOD DELETE CATLG	ignored	MOD DELETE CATLG	ignored	MOD DELETE CATLG	MOD CATLG CATLG	MOD DELETE CATLG
# SORTOUT	MOD DELETE CATLG	ignored	ignored	MOD DELETE CATLG	ignored	ignored	ignored	MOD DELETE CATLG
SYSMAP	ignored	ignored	ignored	MOD CATLG CATLG	ignored	ignored	ignored	ignored
SYSERR	MOD CATLG CATLG	ignored	ignored	MOD CATLG CATLG	ignored	ignored	ignored	ignored
# FILTER # #	ignored	ignored	MOD DELETE CATLG	ignored	ignored	ignored	ignored	ignored

DSNUTILS syntax diagram

For guidance in interpreting syntax diagrams, see “How to read the syntax diagrams” on page 4.

The following syntax diagram shows the SQL CALL statement for invoking utilities as a stored procedure.



DSNUTILS option descriptions

utility-id Specifies a unique identifier for this utility within DB2.

This is an input parameter of type VARCHAR(16).

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

This is an input parameter of type VARCHAR(8).

NO or null Indicates the utility is new, not a restart. There must not be any other utility with the same utility identifier (UID).

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.

utstmt Specifies the utility control statements.

This is an input parameter of type VARCHAR(32704).

retcode Specifies the utility highest return code.

This is an output parameter of type INTEGER.

utility-name Specifies the utility you want to invoke.

This is an input parameter of type VARCHAR(20).

Note: Because you can only specify a single utility here, there is limited dynamic data set allocation support. Specify only a single utility requiring data set allocation in the *utstmt* parameter.

Select the utility name from the following list:

CHECK DATA
CHECK INDEX
CHECK LOB
COPY
DIAGNOSE
LOAD

MERGECOPY
MODIFY RECOVERY
QUIESCE
REBUILD INDEX
RECOVER
REORG INDEX
REORG LOB
REORG TABLESPACE
REPAIR
REPORT RECOVERY
REPORT TABLESPACESET
RUNSTATS INDEX
RUNSTATS TABLESPACE
STOSPACE

recdsn Specifies the cataloged data set name 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*, it will be allocated to the **SYSREC** DDNAME.

This is an input parameter of type VARCHAR(54).

Note: If you specified the **INDDN** parameter for LOAD, the value specified for *ddname* **MUST** be **SYSREC**.

If you specified the **UNLDDN** parameter for REORG TABLESPACE, the value specified for *ddname* **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).

recspace Specifies the number of cylinders to use as the primary space allocation for the *recdsn* data set. The secondary space allocation will be 10% of the primary.

This is an input parameter of type SMALLINT.

discdsn Specifies the cataloged data set name 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 not reloaded. If you specify *discdsn*, it will be allocated to the **SYSDISC** DDNAME.

This is an input parameter of type VARCHAR(54).

Note: If you specified the **DISCARDDN** parameter for LOAD or REORG TABLESPACE, the value specified for *ddname* **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).

discspace Specifies the number of cylinders to use as the primary space allocation for the *discdsn* data set. The secondary space allocation will be 10% of the primary.

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*, it will be allocated to the **SYSPUNCH** DDNAME.

This is an input parameter of type VARCHAR(54).

Note: If you specified the **PUNCHDDN** parameter for REORG TABLESPACE, the value specified for *ddname* **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).

pnchspace Specifies the number of cylinders to use as the primary space allocation for the *pnchdsn* data set. The secondary space allocation will be 10% of the primary.

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 utility or the MERGECOPY utility. It is optional for LOAD and REORG TABLESPACE. If you specify *copydsn1*, it will be allocated to the **SYSCOPY** DDNAME.

This is an input parameter of type VARCHAR(54).

Note: If you specified the **COPYDDN** parameter for COPY, MERGECOPY, LOAD, or REORG TABLESPACE, the value specified for *ddname1* **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).

copyspace1 Specifies the number of cylinders to use as the primary space allocation for the *copydsn1* data set. The secondary space allocation will be 10% of the primary.

This is an input parameter of type SMALLINT.

copydsn2 Specifies the name of the cataloged data set used as a target (output) data set for the backup copy. It is optional for COPY, MERGECOPY, LOAD, and REORG TABLESPACE. If you specify *copydsn2*, it will be allocated to the **SYSCOPY2** DDNAME.

This is an input parameter of type VARCHAR(54).

Note: If you specified the **COPYDDN** parameter for COPY, MERGECOPY, LOAD, or REORG TABLESPACE, the value specified for *ddname2* **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).

copyspace2 Specifies the number of cylinders to use as the primary space allocation for the *copydsn2* data set. The secondary space allocation will be 10% of the primary.

This is an input parameter of type SMALLINT.

rcpydsn1 Specifies the name of the cataloged data set required as a target (output) data set for the remote site primary copy. It is optional for COPY, LOAD, and REORG TABLESPACE. If you specified *rcpydsn1*, it will be allocated to the **SYSRCPY1** DDNAME.

This is an input parameter of type VARCHAR(54).

Note: If you specified the **RECOVERYDDN** parameter for COPY, MERGECOPY, LOAD, or REORG TABLESPACE, the value specified for *ddname1* **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).

rcpyspace1 Specifies the number of cylinders to use as the primary space allocation for the *rcpydsn1* data set. The secondary space allocation will be 10% of the primary.

This is an input parameter of type SMALLINT.

rcpydsn2 Specifies the name of the cataloged data set required as a target (output) data set for the remote site backup copy. It is optional for COPY, LOAD, and REORG TABLESPACE. If you specify *rcpydsn2*, it will be allocated to the **SYSRCPY2** DDNAME.

This is an input parameter of type VARCHAR(54).

Note: If you specified the **RECOVERYDDN** parameter for COPY, MERGECOPY, LOAD, or REORG TABLESPACE, the value specified for *ddname2* **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).

rcpyspace2 Specifies the number of cylinders to use as the primary space allocation for the *rcpydsn2* data set. The secondary space allocation will be 10% of the primary.

This is an input parameter of type SMALLINT.

workdsn1 Specifies the name of the cataloged data set 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 specified the SORTKEYS keyword. It is optional for REBUILD INDEX. If you specify *workdsn1*, it will be allocated to the **SYSUT1** DDNAME.

This is an input parameter of type VARCHAR(54).

Note: If you specified the **WORKDDN** parameter for CHECK DATA, CHECK INDEX, LOAD, REORG INDEX, REORG

TABLESPACE, or REBUILD INDEX, the value specified for *ddname* **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).

workspace1 Specifies the number of cylinders to use as the primary space allocation for the *workdsn1* data set. The secondary space allocation will be 10% of the primary.

This is an input parameter of type SMALLINT.

workdsn2 Specifies the name of the cataloged data set required as a work data set for sort input and output. It is required for CHECK DATA. It is also required if you are using REORG INDEX to reorganize non-unique type 1 indexes. It is required for LOAD or REORG TABLESPACE unless you also specified the SORTKEYS keyword. If you specify *workdsn2*, it will be allocated to the **SORTOUT** DDNAME.

This is an input parameter of type VARCHAR(54).

Note: If you specified the **WORKDDN** parameter for CHECK DATA, LOAD, REORG INDEX, or REORG TABLESPACE, the value specified for *ddname* **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).

workspace2 Specifies the number of cylinders to use as the primary space allocation for the *workdsn2* data set. The secondary space allocation will be 10% of the primary.

This is an input parameter of type SMALLINT.

mapdsn Specifies the name of the cataloged data set required as a work data set for error processing during LOAD with ENFORCE CONSTRAINTS. It is optional for LOAD. If you specify *mapdsn*, it will be allocated to the **SYSMAP** DDNAME.

This is an input parameter of type VARCHAR(54).

Note: If you specified the **MAPDDN** parameter for LOAD, the value specified for *ddname* **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 will be 10% of the primary.

This is an input parameter of type SMALLINT.

errdsn Specifies the name of the cataloged data set required as a work data set for error processing. It is required for CHECK DATA, and is optional for LOAD. If you specify *errdsn*, it will be allocated to the **SYSERR** DDNAME.

This is an input parameter of type VARCHAR(54).

Note: If you specified the **ERRDDN** parameter for CHECK DATA or LOAD, the value specified for *ddname* **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).

errspace Specifies the number of cylinders to use as the primary space allocation for the *errdsn* data set. The secondary space allocation will be 10% of the primary.

This is an input parameter of type SMALLINT.

filtrdsn Specifies the name of the cataloged data set required as a work data set for error processing. It is optional for COPY CONCURRENT. If you specify *filtrdsn*, it will be allocated to the **FILTER** DDNAME.

This is an input parameter of type VARCHAR(54).

Note: If you specified the **FILTERDDN** parameter for COPY, the value specified for *ddname* **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).

filtrspace Specifies the number of cylinders to use as the primary space allocation for the *filtrdsn* data set. The secondary space allocation will be 10% of the primary.

This is an input parameter of type SMALLINT.

Modifying the WLM-established address space

Add SYSIN and SYSPRINT to the JCL procedure for starting the WLM-established address space, in which DSNUTILS runs. You must allocate SYSIN and SYSPRINT in the procedure to temporarily store utility input statements and utility output messages.

Sample program for calling DSNUTILS

Example program DSNTEJ6U in SDSNSAMP shows sample JCL for preparing and executing DSN8EPU. Example program DSN8EPU in SDSNSAMP is a PL/I program which shows creating and binding the DSNUTILS stored procedure to run a utility.

DSNUTILS output

DB2 creates the result set according to the DECLARE statement shown on page on page 517.

Output from a successful execution of the DSNTEJ6U sample job or an equivalent job lists the parameters specified followed by the messages generated by the DB2 DIAGNOSE DISPLAY MEPL utility.

Appendix C. Resetting an advisory or restrictive status

DB2 sets a restrictive or advisory status on an object to control access and help ensure data integrity. This appendix outlines the restrictive and non-restrictive object states that affect utilities, and the steps required 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:

Auxiliary CHECK pending status

The auxiliary CHECK pending 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 86 for information about resetting the auxiliary CHECK pending status.

Table 86. Resetting auxiliary CHECK pending status

Status	Abbreviation	Object Affected	Corrective Action	Notes
Auxiliary CHECK pending	ACHKP	base table space, LOB 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 the following actions (all three must apply):

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 87 for information about resetting the auxiliary warning status.

Table 87. 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
		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. Alternatively, you can use the REPAIR utility to set NOLOBCHKP. Be aware that using the REPAIR utility to delete invalid LOBs might cause the base table and the index on the auxiliary table to reference invalid LOBs.	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 CHECK pending (ACHKP) status.

CHECK pending status

The CHECK pending 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.

Refer to Table 88 for information about resetting the CHECK pending status.

Table 88. 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 CHECK DATA to clear the respective states.	
		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
		LOB table space	Use the CHECK LOB utility to check the LOB table space. If any errors are found: 1. Correct any defects found in the LOB table space using the REPAIR utility. 2. Run CHECK LOB again to reset the CHECK pending status. 3. See Table 87 on page 528 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 restrictive status indicates that the affected object must be copied.

Refer to Table 89 for information about resetting the COPY pending status.

Table 89. 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 status is set on when a coupling facility crashes with pages that were not externalized. The affected object must be recovered.

Refer to Table 90 for information about resetting the group buffer pool RECOVER pending status.

Table 90. 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 advisory status indicates that the affected object should be copied.

Refer to Table 91 for information about resetting the informational COPY pending status.

Table 91. 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

The REBUILD pending (RBDP) restrictive status indicates that the affected index or index partition is broken and must be rebuilt from the data.

Refer to Table 92 for information about resetting the REBUILD pending status.

Table 92 (Page 1 of 2). Resetting REBUILD pending status

Status	Abbreviation	Object Affected	Corrective Action	Notes
REBUILD pending	RBDP	physical or logical index partition	Run the REBUILD or RECOVER utility on the affected index partition.	1
	RBDP*	logical partitions of nonpartitioning indexes	Run REBUILD INDEX PART or RECOVER utility on the affected logical partitions.	
	PSRBD	nonpartitioning 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.	

Table 92 (Page 2 of 2). Resetting REBUILD pending status

Status	Abbreviation	Object Affected	Corrective Action	Notes
			The following actions also reset the REBUILD pending status: • LOAD REPLACE the table space or partition. • 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 containing 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.	

Notes:

1. The entire nonpartitioning index is inaccessible until you reset the RBDP status.

RECOVER pending status

The RECOVER pending restrictive status indicates that a table space or table space partition is broken and must be recovered.

Refer to Table 93 for information about resetting the RECOVER pending status.

Table 93 (Page 1 of 2). Resetting RECOVER pending status

Status	Abbreviation	Object Affected	Corrective Action	Notes
RECOVER pending	RECP	table space	Run the RECOVER utility on the affected object.	
		Table space partition	Recover the logical partition.	
		Index on the auxiliary table	Correct the RECOVER pending status using one of the following utilities: • REBUILD INDEX • RECOVER INDEX • REORG INDEX SORTDATA	

Table 93 (Page 2 of 2). Resetting RECOVER pending status

Status	Abbreviation	Object Affected	Corrective Action	Notes
		Index space	In a coexistence situation, you might have an index space set in RECP status by a Version 5 subsystem that is visible on a Version 6 subsystem. Likewise, you might have an index space set in RBDP, RBDP*, or PSRBDP status by a Version 6 subsystem that is visible on a Version 5 subsystem. In either situation, run one of the following utilities on the affected index space to reset the pending status: • REBUILD INDEX • RECOVER INDEX • REORG INDEX SORTDATA	
			The following actions also reset the RECOVER pending status: • LOAD REPLACE the table space or partition. • REPAIR SET TABLESPACE or INDEX with NORCVRPEND on table space or partition. Be aware that this does not correct the data inconsistency in the table space or partition. • Start the database containing 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.	

REORG pending status

The REORG pending restrictive status indicates that a table space partition is broken and must be reorganized.

Refer to Table 94 on page 533 for information about resetting the REORG pending status.

Table 94. Resetting REORG pending status

Status	Abbreviation	Object Affected	Corrective Action	Notes
REORG pending	REORP	table space	Perform one of the following actions: • LOAD REPLACE 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 CHECK DATA to clear the respective states. • Run REORG PART*m*:*n* SHRLEVEL NONE.	1,2, 3,4
REORG pending	REORP	partitioned table space	For row lengths <= 32KB: 1. Run REORG TABLESPACE SHRLEVEL NONE SORTDATA. For row lengths > 32KB: 1. Run REORG TABLESPACE UNLOAD ONLY. 2. Run LOAD TABLESPACE FORMAT UNLOAD.	

Notes:

1. Consider running COPY after resetting the REORP status. Be aware that you can only use an image copy that was created before turning off the REORP status if you are performing a point-in-time recovery.
2. You cannot run SELECT, INSERT, DELETE, or UPDATE on data in a table space that is in REORP status; this includes access through a partitioning or nonpartitioning index. The only SQL access that is allowed is DROP TABLESPACE.
3. The START DATABASE ACCESS FORCE command does *not* remove the REORP status from an object.
4. You must allocate a discard data set (SYSDISC) or specify DISCARDDN if the last partition is in REORP.

Restart pending status

The restart pending status is set on if an object has back-out work pending at the end of DB2 restart.

Refer to Table 95 for information about resetting the restart pending status.

Table 95 (Page 1 of 2). 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 back-out work is complete, or until restart is canceled and a conditional restart or cold start is performed in its place. See Section 4 (Volume 1) of *DB2 Administration Guide* for information about the RESTP restrictive status.	1,2,3

Table 95 (Page 2 of 2). Resetting restart pending status

Status	Abbreviation	Object Affected	Corrective Action	Notes

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 RESTP page sets or partitions is not allowed. Any attempt to access a RESTP page set or partition terminates with return code 8.

Appendix D. How to run sample programs DSNTIAUL, DSNTIAD, and DSNTEP2

DB2 provides three 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, 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 32KB 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.

DSNTIAD A sample dynamic SQL program in assembler language. With this program, you can execute any SQL statement that can be executed dynamically, except a SELECT statement.

DSNTEP2 A sample dynamic SQL program 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.

Because these three programs also accept the static SQL statements CONNECT, SET CONNECTION, and RELEASE, you can use the programs to access DB2 tables at remote locations.

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, you must precompile, compile, link and bind it. You need to bind the object code version of DSNTEP2 before you can use it. Usually, your system administrator prepares the programs as part of the installation process. Table 96 indicates which installation job prepares each sample program. All installation jobs are in data set DSN610.SDSNSAMP.

Table 96. Jobs that prepare DSNTIAUL, DSNTIAD, and DSNTEP2

Program Name	Program Preparation Job
DSNTIAUL	DSNTEJ2A
DSNTIAD	DSNTIJTM
DSNTEP2 (source)	DSNTEJ1P
DSNTEP2 (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 97 on page 536 lists the load module name and plan name you must specify, and the parameters you can specify when you run each program. See the following sections for the meaning of each parameter.

Table 97. DSN RUN option values for DSNTIAUL, DSNTIAD, and DSNTEP2

Program Name	Load Module	Plan	Parameters
DSNTIAUL	DSNTIAUL	DSNTIB61	SQL
DSNTIAD	DSNTIAD	DSNTIA61	RC0 SQLTERM(*termchar*)
DSNTEP2	DSNTEP2	DSNTEP61	ALIGN(MID) or ALIGN(LHS) MAXSEL(*n*) NOMIXED or MIXED SQLTERM(*termchar*)

The remainder of this appendix contains the following information about running each program:

- Descriptions of the input parameters
- Data sets you must allocate before you run the program
- Return codes from the program
- Examples of invocation

See the sample jobs listed in Table 96 on page 535 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.

DSNTIAUL parameters: DSNTIAUL accepts one parameter, SQL. If you specify this parameter, your input data set contains one or more complete SQL statements, each of which ends with a semi-colon. 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. The maximum length for a statement is 32765 bytes. 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.

If you do not specify the SQL parameter, your input data set must contain one or more single-line statements (without a semi-colon) 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 semi-colon. 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.

For both input formats, you can specify SELECT statements that join two or more tables or select specific columns from a table. If you specify columns, you will need to modify the LOAD statement that DSNTIAUL generates.

DSNTIAUL data sets:

Data Set	Description
SYSIN	Input data set. See *DSNTIAUL parameters* for information on the contents of the input data. 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 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:

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.
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 you want to unload the rows for department D01 from the project table. You can fit the table specification on one line, and you do not want to execute any non-SELECT statements, so you do not need the SQL parameter. Your invocation looks like this:

```
//UNLOAD  EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIB61) -
       LIB('DSN610.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 *
DSN8610.PROJ WHERE DEPTNO='D01'
```

Figure 36. 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 like this:

```
//UNLOAD  EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIB61) -
       LIB('DSN610.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 *
DSN8610.DEPT
```

Figure 37. 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

For these activities, you must specify the SQL parameter when you run DSNTIAUL. Your DSNTIAUL invocation looks like this:

```
//UNLOAD  EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIB61) PARMS('SQL') -
       LIB('DSN610.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 DSN8610.EMP IN SHARE MODE;
LOCK TABLE DSN8610.PROJ IN SHARE MODE;
SELECT * FROM DSN8610.PROJ;
SELECT * FROM DSN8610.EMP
  WHERE WORKDEPT LIKE 'D%'
  ORDER BY EMPNO;
```

Figure 38. 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 98 on page 540. SQLTERM(;) is the default.

Table 98. Invalid special characters for the SQL terminator

Name	Character	Hexadecimal Representation
blank		X'40'
comma	,	X'5E'
double quote	"	X'7F'
left parenthesis	(	X'4D'
right parenthesis	)	X'5D'
single quote	'	X'7D'
underscore	_	X'6D'

Use a character other than a semicolon if you plan to execute a statement that contains embedded semicolons. For example, suppose you 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#
```

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 semi-colon. 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 and the block size to 1210.

Define all data sets as sequential data sets.

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 32 760 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 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 this:

```
//RUNTIAD EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA61) PARMS('RC0') -
       LIB('DSN610.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
UPDATE DSN8610.PROJ SET DEPTNO='J01' WHERE DEPTNO='A01';
UPDATE DSN8610.PROJ SET DEPTNO='J02' WHERE DEPTNO='A02';
⋮
UPDATE DSN8610.PROJ SET DEPTNO='J20' WHERE DEPTNO='A20';
```

Figure 39. DSNTIAD Invocation with the RC0 Parameter

Running DSNTEP2

This section contains information that you need when you run DSNTEP2, including parameters, data sets, return codes, and invocation examples.

DSNTEP2 parameters:

Parameter
Description

ALIGN(MID) or ALIGN(LHS)
If you want your DSNTEP2 output centered, specify ALIGN(MID). If you want the output left-aligned, choose ALIGN(LHS). The default is ALIGN(MID).

MAXSEL(*n*)
Specify MAXSEL(*n*) to limit the number of rows that DSNTEP2 returns from a SELECT statement. *n* is an integer between 0 and 32768. If you do not specify MAXSEL(*n*), DSNTEP2 returns all rows in the result table.

NOMIXED or MIXED
If your input to DSNTEP2 contains any DBCS characters, specify MIXED. If your input contains no DBCS characters, specify NOMIXED. The default is NOMIXED.

SQLTERM(*termchar*)
Specify this parameter to indicate the character that you use to end each SQL statement. You can use any character *except* one of those listed in Table 98 on page 540. SQLTERM(;) is the default.

Use a character other than a semicolon if you plan to execute a statement that contains embedded semicolons. For example, suppose 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#
```

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. For 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 discussion of the SYSIN data set for more information on the --#SET control statement.

DSNTEP2 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 semi-colon. A statement can span multiple lines, but DSNTEP2 reads only the first 72 bytes of each line. You can enter comments in DSNTEP2 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 and control statements cannot span lines. You can enter a number of control statements in the DSNTEP2 input data set. Those control statements are of the form `--#SET control-option value` The control options are: **TERMINATOR** The SQL statement terminator. *value* is any single-byte character other than one of those listed in Table 98 on

page 540. The default is the value of the SQLTERM parameter.

ROWS_FETCH
The number of rows 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 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.

SYSPRINT Output data set. DSNTEP2 writes informational and error messages in this data set. DSNTEP2 writes output records of no more than 133 bytes.

Define all data sets as sequential data sets.

DSNTEP2 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 32 760 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 you want to use DSNTEP2 to execute SQL SELECT statements that might contain DBCS characters. You also want your output left-aligned. Your invocation looks like this:

```
//RUNTEP2 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
 RUN  PROGRAM(DSNTEP2) PLAN(DSNTEP61) PARMS('/ALIGN(LHS) MIXED') -
       LIB('DSN610.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
SELECT * FROM DSN8610.PROJ;
```

Figure 40. DSNTEP2 Invocation with the ALIGN(LHS) and MIXED parameters

Appendix E. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, 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 publication 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 as your own risk.

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
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
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 information 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 illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Programming interface information

This book is intended to help you to use DB2 for OS/390 utilities.

This book also documents General-use Programming Interface and Associated Guidance Information and Product-sensitive Programming Interface and Associated Guidance Information provided by IBM DATABASE 2 Universal Database Server for OS/390 (DB2 for OS/390).

General-use programming interfaces allow the customer to write programs that obtain the services of DB2 for OS/390.

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

The following terms are trademarks of the International Business Machines Corporation in the United States, or other countries, or both:

AIX
APL2
AS/400
BookManager
CICS
CICS/ESA
DATABASE 2
DataHub
DataPropagator
DB2
DB2 Connect
DB2 Universal Database
DFSMS
DFSMSdfp
DFSMSdss
DFSMShsm
DFSMS/MVS
DFSORT
DRDA
DXT
eNetwork
Enterprise System/3090
Enterprise System/9000
ESA/390
IBM
IMS
IMS/ESA
Language Environment
MVS/DFP
MVS/ESA
Net.Data
OS/2
OS/390
Parallel Sysplex
QMF
RACF
RAMAC
RETAIN
RMF
SQL/DS
System/390
VTAM
3090

Throughout the library, *COBOL* is used to represent OS/VS COBOL, VS COBOL II, IBM COBOL, and COBOL/370 programming languages.

Tivoli™ and NetView™ are trademarks of Tivoli Systems Inc. in the United States, or other countries, or both.

The following terms are trademarks of other companies as follows:

- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft™ Corporation in the United States and/or other countries.
- UNIX® is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

Other company, product, or service names may be trademarks or service marks of others.

Glossary

The following terms and abbreviations are defined as they are used in the DB2 library. If you do not find the term you are looking for, refer to the index or to *IBM Dictionary of Computing*.

A

abend. Abnormal end of task.

abend reason code. A 4-byte hexadecimal code that uniquely identifies a problem with DB2. A complete list of DB2 abend reason codes and their explanations is contained in *DB2 Messages and Codes*.

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.

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*.

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 independently of the allied agent, such as prefetch processing, deferred writes, and service tasks.

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.

ambiguous cursor. A database cursor that is not defined with the FOR FETCH ONLY clause or the FOR UPDATE OF clause, is not defined on a read-only result table, is not the target of a WHERE CURRENT clause on an SQL UPDATE or DELETE statement, and is in a plan or package that contains either PREPARE or EXECUTE IMMEDIATE SQL statements.

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.

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 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 server (AS). See *server*.

archive log. The portion of the DB2 log that contains log records that have been copied from the active log.

AS. Application server. See *server*.

ASCII. An encoding scheme that is used to represent strings in many environments, typically on PCs and workstations. Contrast with *EBCDIC*.

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 are allowed. It can represent an individual, an organizational group, or a function, but DB2 does not determine this representation.

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

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 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.

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, where the size of the value ranges from 0 bytes to 2 GB - 1. Such a string does not have an associated CCSID.

bind. The process by which the output from the DB2 precompiler is converted to a usable control structure (which is called a package or an application plan). During the process, access paths to the data are selected and some authorization checking is performed.

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, because they could not be bound during the bind process, and VALIDATE(RUN) was specified.
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.

BLOB. Binary large object.

BMP. Batch Message Processing (IMS).

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 function. A function that DB2 supplies. Contrast with *user-defined function*.

C

CAF. Call attachment facility.

call attachment facility (CAF). A DB2 attachment facility for application programs that run in TSO or MVS batch. The CAF is an alternative to the DSN command processor and provides greater control over the execution environment.

cascade delete. The way in which DB2 enforces referential constraints when it deletes all descendent rows of a deleted parent row.

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.

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.

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- and double-byte characters.

CHECK clause. An extension to the SQL CREATE TABLE and SQL ALTER TABLE statements that specifies a table check constraint. See also *table check constraint*.

check constraint. See *table check constraint*.

check integrity. The condition that exists when each row in a table conforms to the table check constraints that are defined on that table. Maintaining check integrity requires DB2 to enforce table 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 some SQL statements because of rows that violate referential constraints, table 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.

CI. Control interval.

CICS. Represents (in this publication) one of the following products:

CICS Transaction Server for OS/390: Customer Information Control Center Transaction Server for OS/390
CICS/ESA: Customer Information Control System/Enterprise Systems Architecture
CICS/MVS: Customer Information Control System/Multiple Virtual Storage

CICS attachment facility. A DB2 subcomponent that uses the MVS 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:

Cursor stability (CS)
Repeatable read (RR)
Write

claim count. A count of the number of agents that are accessing an object.

clause. In SQL, a distinct part of a statement, such as a SELECT clause or a WHERE clause.

CLIST. Command list. A language for performing TSO tasks.

CLOB. Character large object.

clustering index. An index that determines how rows are physically ordered in a table space.

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.

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. An SQL operation that derives its result from a collection of values across one or more rows. Contrast with *scalar function*.

command. A DB2 operator command or a DSN subcommand. A command is distinct from an SQL statement.

command recognition character (CRC). A character that permits an MVS console operator or an IMS subsystem user to route DB2 commands to specific DB2 subsystems.

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 multi-site update process that requests all participants to commit the effects of the logical unit of work.

communications database (CDB). A set of tables in the DB2 catalog that are used to establish conversations with remote database management systems.

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 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*, *table check constraint*, and *uniqueness constraint*.

control interval (CI). A fixed-length area or direct access storage in which VSAM stores records and creates distributed free space. Also, in a key-sequenced data set or file, the set of records pointed to by an entry in the sequence-set index record. The control interval is the unit of information that VSAM transmits to or from direct access storage. A control interval always includes an integral number of physical records.

control interval definition field (CIDF). In VSAM, a field 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.

correlation ID. An identifier that is associated with a specific thread. In TSO, it is either an authorization ID or the job name.

CRC. Command recognition character.

CRCR. Conditional restart control record. See also *conditional restart*.

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*.

CT. Cursor table.

current data. Data within a host structure that is current with (identical to) the data within the base table.

current status rebuild. The second phase of restart processing during which the status of the subsystem is reconstructed from information on the log.

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

DASD. Direct access storage device.

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 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, and relationships.

database management system (DBMS). A software system that controls the creation, organization, and modification of a database and the access to the data 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 2 Interactive (DB2I). The DB2 facility that provides for the execution of SQL statements, DB2 (operator) commands, programmer commands, and utility invocation.

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 Language/I (DL/I). The IMS data manipulation language; a common high-level interface between a user application and IMS.

data partition. A VSAM data set that is contained within a partitioned table space.

data type. An attribute of columns, literals, host variables, special registers, and the results of functions and expressions.

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.

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 that contain descriptions of DB2 objects, such as tables, views, and indexes.

DB2 command. An instruction to the DB2 subsystem allowing a user 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. DATABASE 2 Interactive.

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 only be used 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.

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 (_).

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 *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.

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.

direct access storage device (DASD). A device in which access time is independent of the location of the data.

directory. The DB2 system database that contains internal objects such as database descriptors and skeleton cursor tables.

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 facility (DDF). A set of DB2 components through which DB2 communicates with another RDBMS.

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.

DL/I. Data Language/I.

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, double-byte character large object 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 and therefore requires special hardware to be displayed or printed. Contrast with *single-byte character set*.

double-precision floating point number. A 64-bit approximate representation of a real number.

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. A method of accessing distributed data by which you can connect to another location, using an SQL statement, to execute packages that have been previously bound at that location. The SQL CONNECT or three-part name statement is used to identify application servers, and SQL statements are executed using packages that were previously bound at those servers. 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 *date duration*, *labeled duration*, and *time duration*.

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.

E

EBCDIC. Extended binary coded decimal interchange code. An encoding scheme that is used to represent character data in the OS/390, MVS, VM, VSE, and OS/400® environments. Contrast with *ASCII*.

EDM pool. A pool of main storage that is used for database descriptors, application plans, authorization cache, application packages, and dynamic statement caching.

embedded SQL. SQL statements that are coded within an application program. See *static SQL*.

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.

EUR. IBM European Standards.

exception table. A table that holds rows that violate referential constraints or table 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 *shared lock*.

executable statement. An SQL statement that can be embedded in an application program, dynamically prepared and executed, or issued interactively.

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.

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* and *built-in function*.

F

fallback. The process of returning to a previous release of DB2 after attempting or completing migration to a current release.

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.

fixed-length string. A character or graphic string whose length is specified and cannot be changed. Contrast with *varying-length string*.

foreign key. A key that is specified in the definition of a referential constraint. Because of the foreign key, the table is a dependent table. The key must have the same number of columns, with the same descriptions, as the primary key of the parent table.

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.

function. A specific purpose of an entity or its characteristic action such as a column function or scalar function. (See also *column function* and *scalar function*.)

Functions can be user-defined, built-in, or generated by DB2. (See *built-in function*, *cast function*, *external function*, *sourced function*, and *user-defined function*.)

G

GB. Gigabyte (1 073 741 824 bytes).

GBP. Group buffer pool.

generalized trace facility (GTF). An MVS service program that records significant system events such as I/O interrupts, SVC interrupts, program interrupts, or external interrupts.

getpage. An operation in which DB2 accesses a data page.

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*.

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.

GTF. Generalized trace facility.

H

help panel. A screen of information presenting tutorial text to assist a user at the terminal.

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.

HSM. Hierarchical storage manager.

I

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.

identify. A request that an attachment service program in an address space that is separate from DB2 issues via the MVS 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 guaranteed-unique numeric value for each row that is inserted into the table. Identity columns are defined with the AS IDENTITY clause. 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.

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).

IMS. Information Management System.

IMS attachment facility. A DB2 subcomponent that uses MVS subsystem interface (SSI) protocols and cross-memory linkage to process requests from IMS to DB2 and to coordinate resource commitment.

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*.

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 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.

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*.

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).

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.

internal resource lock manager (IRLM). An MVS subsystem that DB2 uses to control communication and database locking.

IRLM. Internal resource lock manager.

ISO. International Standards Organization.

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.

J

Japanese Industrial Standards Committee (JISC). An organization that issues standards for coding character sets.

JCL. Job control language.

JES. MVS 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.

K

KB. Kilobyte (1024 bytes).

key. A column or an ordered collection of columns identified in the description of a table, index, or referential constraint.

key-sequenced data set (KSDS). A VSAM file or data set whose records are loaded in key sequence and controlled by an index.

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*.

leaf page. A page that contains pairs of keys and RIDs and that points to actual data. Contrast with *nonleaf page*.

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.

L-lock. Logical lock.

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 table space. A table space 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*.

local subsystem. The unique RDBMS to which the user or application program is directly connected (in the case of DB2, by one of the DB2 attachment facilities).

location name. The name by which DB2 refers to a particular DB2 subsystem in a network of subsystems. Contrast with *LU name*.

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 level.

lock size. The amount of data 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.

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.

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 *P-lock*.

logical unit. 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 sequence number (LRSN). A number that DB2 generates and associates with each log record. DB2 also uses the LRSN for page versioning. The LRSNs that a particular DB2 data sharing group generates form a strictly increasing sequence for each DB2 log and a strictly increasing sequence for each page across the DB2 group.

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.

LRH. Log record header.

LRSN. Log record sequence number.

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 between the RIDs of data records in the original copy and in the shadow copy. This table is created by the user.

MB. Megabyte (1 048 576 bytes).

menu. A displayed list of available functions for selection by the operator. A menu is sometimes called a *menu panel*.

migration. The process of converting a DB2 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 you created on the previous release.

mixed data string. A character string that can contain both single-byte and double-byte characters.

MPP. Message processing program (IMS).

MTO. Master terminal operator.

multi-site update. Distributed relational database processing in which data is updated in more than one location within a single unit of work.

MVS. Multiple Virtual Storage.

MVS/ESA. Multiple Virtual Storage/Enterprise Systems Architecture.

MVS/XA. Multiple Virtual Storage/Extended Architecture.

N

network identifier (NID). The network ID that is assigned by IMS or CICS, or if the connection type is RRSAF, the OS/390 RRS Unit of Recovery ID (URID).

NID. Network ID.

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.

NUL. In C, a single character that denotes the end of the string.

null. A special value that indicates the absence of information.

NUL-terminated host variable. A varying-length host variable in which the end of the data is indicated by the presence of a NUL terminator.

NUL terminator. In C, the value that indicates the end of a string. For character strings, the NUL terminator is X'00'.

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.

OBID. Data object identifier.

OS/390. Operating System/390.

P

package. An object containing a set of SQL statements that have been bound statically 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 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.

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 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.

partner logical unit. An access point in the SNA network that is connected to the local DB2 subsystem by way of a VTAM conversation.

piece. A data set of a nonpartitioned page set.

plan. See *application plan*.

plan allocation. The process of allocating DB2 resources to a plan in preparation to execute it.

plan name. The name of an application plan.

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*.

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.

prefix. A code at the beginning of a message or record.

prepare. The first phase of a two-phase commit process in which all participants are requested to prepare for commit.

primary authorization ID. The authorization ID used to identify the application process to DB2.

primary index. An index that enforces the uniqueness of a primary key.

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 term includes:

> **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 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.

protected conversation. A VTAM conversation that supports two-phase commit flows.

PSRCP. Page set recovery pending.

Q

QMF. Query Management Facility.

query. A component of certain SQL statements that specifies a result table.

R

RACF. Resource Access Control Facility.

RBA. Relative byte address.

RCT. Resource control table (CICS attachment facility).

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.

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 stored as a record. Compare with *row ID*.

record identifier (RID) pool. An area of main storage above the 16-MB line that is reserved for sorting record identifiers during list prefetch processing.

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 pending (RECP). A condition that prevents SQL access to a table space that needs to be recovered.

RECP. Recovery pending.

redo. A state of a unit of recovery that indicates that changes are to be reapplied to the DASD media to ensure data integrity.

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 condition that exists when all intended references from data in one column of a table to data in another column of the same or a different table are valid. Maintaining referential integrity requires that DB2 enforce referential constraints on all LOAD, RECOVER, INSERT, UPDATE, and DELETE operations.

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.

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 subsystem. Any RDBMS, 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 MVS system.

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 referenced by the program cannot be changed by other programs until the program reaches a commit point.

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.

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 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 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. A site-defined table that specifies the limits to be enforced by the resource limit facility.

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 table. The set of rows that are specified by a SELECT statement.

RID. Record identifier.

RID pool. Record identifier pool.

RLF. Resource limit facility.

RMID. Resource manager identifier.

RO. Read-only access.

rollback. The process of restoring data changed by SQL statements to the state at its last commit point. All locks are freed. Contrast with *commit*.

ROWID. Row identifier.

row identifier (ROWID). A value that uniquely identifies a row. This value is stored with the row and never changes.

RS. Read stability.

S

SBCS. Single-byte character set.

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 *column function*.

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.

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.

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.

server. A functional unit that provides services to one or more clients over a network. In the DB2 environment, a server is the target for a request from a remote RDBMS and is the RDBMS that provides the data. A server is sometimes also called an *application server (AS)*.

session. A link between two nodes in a VTAM network.

shared 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 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*.

SMF. System management facility.

SMS. Storage Management Subsystem.

SNA. Systems Network Architecture.

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 *external function* and *built-in function*.

source program. A set of host language statements and SQL statements that is processed by an SQL precompiler.

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.

SQL communication area (SQLCA). A structure that is used to provide an application program with information about the execution of its SQL statements.

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 Processor Using File Input (SPUFI). SQL Processor Using File Input. 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 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 (").

SQLCA. SQL communication area.

SQLDA. SQL descriptor area.

SQL/DS. Structured Query Language/Data System. This product is now obsolete and has been replaced by DB2 for VSE & VM.

SSI. Subsystem interface (MVS).

SSM. Subsystem member.

stand-alone. An attribute of a program that means it is capable of executing separately from DB2, without using DB2 services.

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 DASD volumes on which DB2 data can be stored.

string. See *character string* or *graphic string*.

Structured Query Language (SQL). A standardized language for defining and manipulating data in a relational database.

subsystem. A distinct instance of a relational database management system (RDBMS).

sync point. See *commit point*.

synonym. In SQL, an alternative name for a table or view. Synonyms can only be used to refer to objects at the subsystem in which the synonym is defined.

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.

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.

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 check constraint. A user-defined constraint that specifies the values that specific columns of a base table can contain.

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 control block that is used to communicate information about tasks within an address space that are connected to DB2. An address space can support many task connections (as many as one per task), but only one address space connection. See also *address space connection*.

TCB. Task control block (MVS).

temporary table. A table that holds temporary data; for example, temporary tables are useful for holding or sorting intermediate results from queries that contain a large number of rows. The two kinds 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*.

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*.

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.

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.

trace. A DB2 facility that provides the ability to monitor and collect DB2 monitoring, auditing, performance, accounting, statistics, and serviceability (global) data.

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.

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 6, type 1 indexes are no longer supported.

type 2 indexes. Indexes that are created on a release of DB2 after Version 5 or that are specified as type 2 indexes in Version 4 or Version 5.

U

UDF. User-defined function.

UDT. User-defined data type. In DB2 for OS/390, the term *distinct type* is used instead of user-defined function.

uncommitted read (UR). The isolation level that allows an application to read uncommitted data.

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.

unique index. An index which ensures that no identical key values are stored in a table.

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.

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 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 *multi-site update* operation, a single unit of work can include several *units of recovery*. Contrast with *unit of recovery*.

UR. Uncommitted read.

URID (unit of recovery ID). 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.

user-defined data type (UDT). See *distinct type*.

user-defined function (UDF). A function that is defined to DB2 using the CREATE FUNCTION statement and that can be referenced thereafter in SQL statements. A user-defined function can be either an *external function* or a *sourced function*. Contrast with *built-in function*.

UT. Utility-only access.

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*.

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.

Virtual Storage Access Method (VSAM). An access method for direct or sequential processing of fixed- and varying-length records on direct access 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.

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in an SNA network.

VSAM. Virtual storage access method.

VTAM. Virtual Telecommunication Access Method (MVS).

W

warm start. The normal DB2 restart process, which involves reading and processing log records so that data under the control of DB2 is consistent. Contrast with *cold start*.

Bibliography

DB2 Universal Database Server for OS/390 Version 6 Product Libraries:

DB2 Universal Database for OS/390

- *DB2 Administration Guide, SC26-9003*
- *DB2 Application Programming and SQL Guide, SC26-9004*
- *DB2 Application Programming Guide and Reference for Java™, SC26-9018*
- *DB2 ODBC Guide and Reference, SC26-9005*
- *DB2 Command Reference, SC26-9006*
- *DB2 Data Sharing: Planning and Administration, SC26-9007*
- *DB2 Data Sharing Quick Reference Card, SX26-3843*
- *DB2 Diagnosis Guide and Reference, LY36-3736*
- *DB2 Diagnostic Quick Reference Card, LY36-3737*
- *DB2 Image, Audio, and Video Extenders Administration and Programming, SC26-9650*
- *DB2 Installation Guide, GC26-9008*
- *DB2 Licensed Program Specifications, GC26-9009*
- *DB2 Messages and Codes, GC26-9011*
- *DB2 Master Index, SC26-9010*
- *DB2 Reference for Remote DRDA Requesters and Servers, SC26-9012*
- *DB2 Reference Summary, SX26-3844*
- *DB2 Release Planning Guide, SC26-9013*
- *DB2 SQL Reference, SC26-9014*
- *DB2 Text Extender Administration and Programming, SC26-9651*
- *DB2 Utility Guide and Reference, SC26-9015*
- *DB2 What's New? GC26-9017*
- *DB2 Program Directory, GI10-8182*

DB2 Administration Tool

- *DB2 Administration Tool for OS/390 User's Guide, SC26-9847*

DB2 Buffer Pool Tool

- *DB2 Buffer Pool Tool for OS/390 User's Guide and Reference, SC26-9306*

DB2 DataPropagator

- *DB2 Replication Guide and Reference, SC26-9642*

Net.Data for OS/390

The following books are available at
http://www.ibm.com/software/net.data/library.html:

- *Net.Data Library: Administration and Programming Guide for OS/390*
- *Net.Data Library: Language Environment Interface Reference*
- *Net.Data Library: Messages and Codes*
- *Net.Data Library: Reference*

DB2 PM for OS/390

- *DB2 PM for OS/390 Batch User's Guide, SC26-9167*
- *DB2 PM for OS/390 Command Reference, SC26-9166*
- *DB2 PM for OS/390 General Information, GC26-9172*
- *DB2 PM for OS/390 Installation and Customization, SC26-9171*
- *DB2 PM for OS/390 Messages, SC26-9169*
- *DB2 PM for OS/390 Online Monitor User's Guide, SC26-9168*
- *DB2 PM for OS/390 Report Reference Volume 1, SC26-9164*
- *DB2 PM for OS/390 Report Reference Volume 2, SC26-9165*
- *DB2 PM for OS/390 Using the Workstation Online Monitor, SC26-9170*
- *DB2 PM for OS/390 Program Directory, GI10-8183*

Query Management Facility

- *Query Management Facility: Developing QMF Applications, SC26-9579*
- *Query Management Facility: Getting Started with QMF on Windows, SC26-9582*
- *Query Management Facility: High Peformance Option User's Guide for OS/390, SC26-9581*
- *Query Management Facility: Installing and Managing QMF on OS/390, GC26-9575*
- *Query Management Facility: Installing and Managing QMF on Windows, GC26-9583*
- *Query Management Facility: Introducing QMF, GC26-9576*
- *Query Management Facility: Messages and Codes, GC26-9580*
- *Query Management Facility: Reference, SC26-9577*
- *Query Management Facility: Using QMF, SC26-9578*

Ada/370

- *IBM Ada/370 Language Reference, SC09-1297*
- *IBM Ada/370 Programmer's Guide, SC09-1414*
- *IBM Ada/370 SQL Module Processor for DB2 Database Manager User's Guide, SC09-1450*

APL2

- *APL2 Programming Guide, SH21-1072*
- *APL2 Programming: Language Reference, SH21-1061*
- *APL2 Programming: Using Structured Query Language (SQL), SH21-1057*

AS/400

- *DB2 for OS/400 SQL Programming, SC41-4611*
- *DB2 for OS/400 SQL Reference, SC41-4612*

BASIC

- *IBM BASIC/MVS Language Reference, GC26-4026*
- *IBM BASIC/MVS Programming Guide, SC26-4027*

BookManager READ/MVS

- *BookManager READ/MVS V1R3: Installation Planning & Customization, SC38-2035*

C/370

- *IBM SAA AD/Cycle C/370 Programming Guide, SC09-1841*
- *IBM SAA AD/Cycle C/370 Programming Guide for Language Environment/370, SC09-1840*
- *IBM SAA AD/Cycle C/370 User's Guide, SC09-1763*
- *SAA CPI C Reference, SC09-1308*

Character Data Representation Architecture

- *Character Data Representation Architecture Overview, GC09-2207*
- *Character Data Representation Architecture Reference and Registry, SC09-2190*

CICS/ESA

- *CICS/ESA Application Programming Guide, SC33-1169*
- *CICS for MVS/ESA Application Programming Reference, SC33-1170*
- *CICS for MVS/ESA CICS-RACF Security Guide, SC33-1185*
- *CICS for MVS/ESA CICS-Supplied Transactions, SC33-1168*
- *CICS for MVS/ESA Customization Guide, SC33-1165*
- *CICS for MVS/ESA Data Areas, LY33-6083*
- *CICS for MVS/ESA Installation Guide, SC33-1163*
- *CICS for MVS/ESA Intercommunication Guide, SC33-1181*
- *CICS for MVS/ESA Messages and Codes, GC33-1177*
- *CICS for MVS/ESA Operations and Utilities Guide, SC33-1167*
- *CICS/ESA Performance Guide, SC33-1183*
- *CICS/ESA Problem Determination Guide, SC33-1176*
- *CICS for MVS/ESA Resource Definition Guide, SC33-1166*
- *CICS for MVS/ESA System Definition Guide, SC33-1164*
- *CICS for MVS/ESA System Programming Reference, GC33-1171*

CICS/MVS

- *CICS/MVS Application Programmer's Reference, SC33-0512*
- *CICS/MVS Facilities and Planning Guide, SC33-0504*
- *CICS/MVS Installation Guide, SC33-0506*
- *CICS/MVS Operations Guide, SC33-0510*
- *CICS/MVS Problem Determination Guide, SC33-0516*
- *CICS/MVS Resource Definition (Macro), SC33-0509*
- *CICS/MVS Resource Definition (Online), SC33-0508*

IBM C/C++ for MVS/ESA

- *IBM C/C++ for MVS/ESA Library Reference, SC09-1995*
- *IBM C/C++ for MVS/ESA Programming Guide, SC09-1994*

IBM COBOL

- *IBM COBOL Language Reference, SC26-4769*
- *IBM COBOL for MVS & VM Programming Guide, SC26-4767*

Conversion Guide

- *IMS-DB and DB2 Migration and Coexistence Guide, GH21-1083*

Cooperative Development Environment

- *CoOperative Development Environment/370: Debug Tool, SC09-1623*

Data Extract (DXT)

- *Data Extract Version 2: General Information, GC26-4666*
- *Data Extract Version 2: Planning and Administration Guide, SC26-4631*

DataPropagator NonRelational

- *DataPropagator NonRelational MVS/ESA Administration Guide, SH19-5036*
- *DataPropagator NonRelational MVS/ESA Reference, SH19-5039*

Data Facility Data Set Services

- *Data Facility Data Set Services: User's Guide and Reference, SC26-4388*

Database Design

- *DB2 Design and Development Guide, Gabrielle Wiorkowski and David Kull, Addison Wesley, ISBN 0-20158-049-8*
- *Handbook of Relational Database Design, C. Fleming and B. Von Halle, Addison Wesley, ISBN 0-20111-434-8*

DataHub

- *IBM DataHub General Information, GC26-4874*

DB2 Connect

- *DB2 Connect Enterprise Edition for OS/2 and Windows NT: Quick Beginnings, GC09-2828*
- *DB2 Connect Personal Edition Quick Beginnings, GC09-2830*
- *DB2 Connect User's Guide, SC09-2838*

DB2 Server for VSE & VM

- *DB2 Server for VM: DBS Utility, SC09-2394*
- *DB2 Server for VSE: DBS Utility, SC09-2395*

DB2 Universal Database (UDB)

- *DB2 UDB Administration Guide Volume 1: Design and Implementation, SC09-2839*
- *DB2 UDB Administration Guide Volume 2: Performance, SC09-2840*
- *DB2 UDB Administrative API Reference, SC09-2841*
- *DB2 UDB Application Building Guide, SC09-2842*
- *DB2 UDB Application Development Guide, SC09-2845*
- *DB2 UDB Call Level Interface Guide and Reference, SC09-2843*
- *DB2 UDB SQL Getting Started, SC09-2856*
- *DB2 UDB SQL Reference Volume 1, SC09-2847*
- *DB2 UDB SQL Reference Volume 2, SC09-2848*

Device Support Facilities

- *Device Support Facilities User's Guide and Reference, GC35-0033*

DFSMS/MVS

- *DFSMS/MVS: Access Method Services for the Integrated Catalog, SC26-4906*
- *DFSMS/MVS: Access Method Services for VSAM Catalogs, SC26-4905*
- *DFSMS/MVS: Administration Reference for DFSMSdss, SC26-4929*
- *DFSMS/MVS: DFSMShsm Managing Your Own Data, SH21-1077*
- *DFSMS/MVS: Diagnosis Reference for DFSMSdfp, LY27-9606*
- *DFSMS/MVS Storage Management Library: Implementing System-Managed Storage, SC26–3123*
- *DFSMS/MVS: Macro Instructions for Data Sets, SC26-4913*
- *DFSMS/MVS: Managing Catalogs, SC26-4914*
- *DFSMS/MVS: Program Management, SC26-4916*
- *DFSMS/MVS: Storage Administration Reference for DFSMSdfp, SC26-4920*
- *DFSMS/MVS: Using Advanced Services, SC26-4921*
- *DFSMS/MVS: Utilities, SC26-4926*
- *MVS/DFP: Using Data Sets, SC26-4749*

DFSORT

- *DFSORT Application Programming: Guide, SC33-4035*

Distributed Relational Database

- *Data Stream and OPA Reference, SC31-6806*
- *IBM SQL Reference, SC26-8416*
- *Open Group Technical Standard (the Open Group presently makes the following books available through its Web site at http://www.opengroup.org):*
  - *DRDA Volume 1: Distributed Relational Database Architecture (DRDA), ISBN 1-85912-295-7*
  - *DRDA Version 2 Volume 2: Formatted Data Object Content Architecture, available only on Web*
  - *DRDA Volume 3: Distributed Database Management (DDM) Architecture, ISBN 1-85912-206-X*

Domain Name System

- *DNS and BIND, Third Edition, Paul Albitz and Cricket Liu, O'Reilly, SR23-8771*

Education

- *IBM Dictionary of Computing, McGraw-Hill, ISBN 0-07031-489-6*
- *1999 IBM All-in-One Education and Training Catalog, GR23-8105*

Enterprise System/9000 and Enterprise System/3090

- *Enterprise System/9000 and Enterprise System/3090 Processor Resource/System Manager Planning Guide, GA22-7123*

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*

Parallel Sysplex Library

- *OS/390 Parallel Sysplex Application Migration, GC28-1863*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28-1862*
- *OS/390 Parallel Sysplex Overview: An Introduction to Data Sharing and Parallelism, GC28-1860*
- *OS/390 Parallel Sysplex Systems Management, GC28-1861*
- *OS/390 Parallel Sysplex Test Report, GC28-1963*
- *System/390 9672/9674 System Overview, GA22-7148*

ICSF/MVS

- *ICSF/MVS General Information, GC23-0093*

IMS/ESA

- *IMS Batch Terminal Simulator General Information, GH20-5522*
- *IMS/ESA Administration Guide: System, SC26-8013*
- *IMS/ESA Administration Guide: Transaction Manager, SC26-8731*
- *IMS/ESA Application Programming: Database Manager, SC26-8727*
- *IMS/ESA Application Programming: Design Guide, SC26-8016*
- *IMS/ESA Application Programming: Transaction Manager, SC26-8729*
- *IMS/ESA Customization Guide, SC26-8020*
- *IMS/ESA Installation Volume 1: Installation and Verification, SC26-8023*
- *IMS/ESA Installation Volume 2: System Definition and Tailoring, SC26-8024*
- *IMS/ESA Messages and Codes, SC26-8028*
- *IMS/ESA Operator's Reference, SC26-8030*
- *IMS/ESA Utilities Reference: System, SC26-8035*

ISPF

- *ISPF V4 Dialog Developer's Guide and Reference, SC34-4486*
- *ISPF V4 Messages and Codes, SC34-4450*
- *ISPF V4 Planning and Customizing, SC34-4443*
- *ISPF V4 User's Guide, SC34-4484*

Language Environment

- *Debug Tool User's Guide and Reference, SC09-2137*

National Language Support

- *National Language Support Reference Volume 2, SE09-8002*

NetView

- *NetView Installation and Administration Guide, SC31-8043*
- *NetView User's Guide, SC31-8056*

ODBC

- *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide, Microsoft Press, ISBN 1-55615-658-8*

OS/390

- *OS/390 C/C++ Programming Guide, SC09-2362*
- *OS/390 C/C++ Run-Time Library Reference, SC28-1663*
- *OS/390 C/C++ User's Guide, SC09-2361*
- *OS/390 eNetwork Communications Server: IP Configuration, SC31-8513*
- *OS/390 Hardware Configuration Definition Planning, GC28-1750*
- *OS/390 Information Roadmap, GC28-1727*
- *OS/390 Introduction and Release Guide, GC28-1725*
- *OS/390 JES2 Initialization and Tuning Guide, SC28-1791*
- *OS/390 JES3 Initialization and Tuning Guide, SC28-1802*
- *OS/390 Language Environment for OS/390 & VM Concepts Guide, GC28-1945*
- *OS/390 Language Environment for OS/390 & VM Customization, SC28-1941*
- *OS/390 Language Environment for OS/390 & VM Debugging Guide, SC28-1942*
- *OS/390 Language Environment for OS/390 & VM Programming Guide, SC28-1939*
- *OS/390 Language Environment for OS/390 & VM Programming Reference, SC28-1940*
- *OS/390 MVS Diagnosis: Procedures, LY28-1082*
- *OS/390 MVS Diagnosis: Reference, SY28-1084*
- *OS/390 MVS Diagnosis: Tools and Service Aids, LY28-1085*
- *OS/390 MVS Initialization and Tuning Guide, SC28-1751*
- *OS/390 MVS Initialization and Tuning Reference, SC28-1752*
- *OS/390 MVS Installation Exits, SC28-1753*
- *OS/390 MVS JCL Reference, GC28-1757*
- *OS/390 MVS JCL User's Guide, GC28-1758*
- *OS/390 MVS Planning: Global Resource Serialization, GC28-1759*
- *OS/390 MVS Planning: Operations, GC28-1760*
- *OS/390 MVS Planning: Workload Management, GC28-1761*
- *OS/390 MVS Programming: Assembler Services Guide, GC28-1762*
- *OS/390 MVS Programming: Assembler Services Reference, GC28-1910*
- *OS/390 MVS Programming: Authorized Assembler Services Guide, GC28-1763*
- *OS/390 MVS Programming: Authorized Assembler Services Reference, Volumes 1-4, GC28-1764, GC28-1765, GC28-1766, GC28-1767*
- *OS/390 MVS Programming: Callable Services for High-Level Languages, GC28-1768*
- *OS/390 MVS Programming: Extended Addressability Guide, GC28-1769*

- *OS/390 MVS Programming: Sysplex Services Guide, GC28-1771*
- *OS/390 MVS Programming: Sysplex Services Reference, GC28-1772*
- *OS/390 MVS Programming: Workload Management Services, GC28-1773*
- *OS/390 MVS Routing and Descriptor Codes, GC28-1778*
- *OS/390 MVS Setting Up a Sysplex, GC28-1779*
- *OS/390 MVS System Codes, GC28-1780*
- *OS/390 MVS System Commands, GC28-1781*
- *OS/390 MVS System Messages Volume 1, GC28-1784*
- *OS/390 MVS System Messages Volume 2, GC28-1785*
- *OS/390 MVS System Messages Volume 3, GC28-1786*
- *OS/390 MVS System Messages Volume 4, GC28-1787*
- *OS/390 MVS System Messages Volume 5, GC28-1788*
- *OS/390 MVS Using the Subsystem Interface, SC28-1789*
- *OS/390 Security Server (RACF) Auditor's Guide, SC28-1916*
- *OS/390 Security Server (RACF) Command Language Reference, SC28-1919*
- *OS/390 Security Server (RACF) General User's Guide, SC28-1917*
- *OS/390 Security Server (RACF) Introduction, GC28-1912*
- *OS/390 Security Server (RACF) Macros and Interfaces, SK2T-6700 (OS/390 Collection Kit), SK27-2180 (OS/390 Security Server Information Package)*
- *OS/390 Security Server (RACF) Security Administrator's Guide, SC28-1915*
- *OS/390 Security Server (RACF) System Programmer's Guide, SC28-1913*
- *OS/390 SMP/E Reference, SC28-1806*
- *OS/390 SMP/E User's Guide, SC28-1740*
- *OS/390 RMF User's Guide, SC28-1949*
- *OS/390 TSO/E CLISTS, SC28-1973*
- *OS/390 TSO/E Command Reference, SC28-1969*
- *OS/390 TSO/E Customization, SC28-1965*
- *OS/390 TSO/E Messages, GC28-1978*
- *OS/390 TSO/E Programming Guide, SC28-1970*
- *OS/390 TSO/E Programming Services, SC28-1971*
- *OS/390 TSO/E User's Guide, SC28-1968*
- *OS/390 DCE Administration Guide, SC28-1584*
- *OS/390 DCE Introduction, GC28-1581*
- *OS/390 DCE Messages and Codes, SC28-1591*
- *OS/390 UNIX System Services Command Reference, SC28-1892*
- *OS/390 UNIX System Services Planning, SC28-1890*
- *OS/390 UNIX System Services User's Guide, SC28-1891*
- *OS/390 UNIX System Services Programming: Assembler Callable Services Reference, SC28-1899*

j

PL/I for MVS & VM

- *IBM PL/I MVS & VM Language Reference, SC26-3114*
- *IBM PL/I MVS & VM Programming Guide, SC26-3113*

OS PL/I

- *OS PL/I Programming Language Reference, SC26-4308*
- *OS PL/I Programming Guide, SC26-4307*

Prolog

- *IBM SAA AD/Cycle Prolog/MVS & VM Programmer's Guide, SH19-6892*

Remote Recovery Data Facility

- *Remote Recovery Data Facility Program Description and Operations, LY37-3710*

Storage Management

- *DFSMS/MVS Storage Management Library: Implementing System-Managed Storage, SC26-3123*
- *MVS/ESA Storage Management Library: Leading a Storage Administration Group, SC26-3126*
- *MVS/ESA Storage Management Library: Managing Data, SC26-3124*
- *MVS/ESA Storage Management Library: Managing Storage Groups, SC26-3125*
- *MVS Storage Management Library: Storage Management Subsystem Migration Planning Guide, SC26-4659*

System/370 and System/390

- *ESA/370 Principles of Operation, SA22-7200*
- *ESA/390 Principles of Operation, SA22-7201*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28-1210*

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*

VS COBOL II

- *VS COBOL II Application Programming Guide for MVS and CMS, SC26-4045*
- *VS COBOL II Application Programming: Language Reference, GC26-4047*
- *VS COBOL II Installation and Customization for MVS, SC26-4048*

VS FORTRAN

- *VS FORTRAN Version 2: Language and Library Reference, SC26-4221*
- *VS FORTRAN Version 2: Programming Guide for CMS and MVS, SC26-4222*

VTAM

- *Planning for NetView, NCP, and VTAM, SC31-8063*
- *VTAM for MVS/ESA Diagnosis, LY43-0069*
- *VTAM for MVS/ESA Messages and Codes, SC31-6546*
- *VTAM for MVS/ESA Network Implementation Guide, SC31-6548*
- *VTAM for MVS/ESA Operation, SC31-6549*
- *VTAM for MVS/ESA Programming, SC31-6550*
- *VTAM for MVS/ESA Programming for LU 6.2, SC31-6551*
- *VTAM for MVS/ESA Resource Definition Reference, SC31-6552*

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

How to send your comments

DB2 Universal Database for OS/390
Utility Guide and Reference
Version 6

Publication No. SC26-9015-01

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 for OS/390 documentation. You can use any of the following methods to provide comments.

- Send your comments by e-mail to db2pubs@vnet.ibm.com and include the name of the product, the version number of the product 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, page number, or a help topic title).
- Send your comments from the Web. Visit the DB2 for OS/390 Web site at:

  http://www.ibm.com/software/db2os390

  The Web site has a feedback page that you can use to send comments.
- Complete the readers' comment form at the back of the book and return it by mail, by fax (800-426-7773 for the United States and Canada), or by giving it to an IBM representative.

Readers' Comments

DB2 Universal Database for OS/390
Utility Guide and Reference
Version 6

Publication No. SC26-9015-01

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐
Grammatically correct and consistent	☐	☐	☐	☐	☐
Graphically well designed	☐	☐	☐	☐	☐
Overall satisfaction	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

May we contact you to discuss your comments? ☐ Yes ☐ No

Name

Address

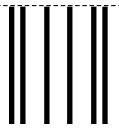
Company or Organization

Phone No.

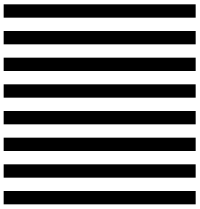
Readers' Comments
SC26-9015-01

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department BWE/H3
PO Box 49023
San Jose, CA 95161-9945

Fold and Tape **Please do not staple** Fold and Tape

SC26-9015-01

Cut or Fold
Along Line

Program Number: 5645-DB2

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-9015-01