DB2 for OS/390
Version 5

Utility Guide and Reference

SC26-8967-02

Note!

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

First Edition (June 1997)

This edition applies to Version 5 of IBM DATABASE 2 Server for OS/390 (DB2 for OS/390), 5655-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.

The technical changes for this edition are summarized under “Summary of Changes to this Book” in the Introduction. Specific changes are indicated by a vertical bar to the left of a change. A vertical bar to the left of a figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not noted.

This softcopy version is based on the printed version 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.

Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter 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.

Licensees of this program who wish to have information about it for the purpose of enabling (1) the exchange of information between independently created programs and other programs (including this one) and (2) the mutual use of the information that has been exchanged, should contact:

IBM Corporation
IBM Corporation
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Programming Interface Information

This book is intended to help you to use DB2 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 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 need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs by 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 IBM Corporation in the United States or other countries or both:

AD/Cycle
APL2
C/370
CICS
CICS/ESA
CICS/MVS
COBOL/370
DATABASE 2
DataPropagator NonRelational
DB2
DFSMS
DFSMS/MVS
DFSMShsm
DFSORT
DRDA
DPROP
IBM
IMS
IMS/ESA
MVS
MVS/ESA
OS/390
QMF
RACF
System/370
System/390
SQL/DS
VTAM

Throughout the library, the DB2 licensed program and a particular DB2 subsystem are each referred to as “DB2.” In each case, the context makes the meaning clear. The term *MVS* is used to represent the MVS/Enterprise Systems Architecture (MVS/ESA); *CICS* is used to represent CICS/MVS and CICS/ESA; *IMS* is used to represent IMS/ESA; *C* and *C language* are used to represent the C/370 programming language. *COBOL* is used to represent OS/VS COBOL, VS COBOL II, IBM COBOL, and COBOL/370 programming languages.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Section 1. Introduction

Chapter 1-1. Introduction to This Book and the DB2 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, starting, stopping, and restarting. Examples are also included for each utility.

Who Should Read This Book

This book is intended for users of DB2 online and stand-alone utilities. It assumes that the reader is already familiar with DB2 for OS/390.

How To Use This Book

It is assumed that you possess an understanding of system administration, database administration, 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:

- "Chapter 1-1. Introduction to This Book and the DB2 Library."
- "Chapter 2-1. Invoking DB2 Online Utilities" on page 2-7.
- "Chapter 3-1. Invoking Stand-Alone Utilities" on page 3-5.
- "Limits in DB2 for OS/390" on page X-3.

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.

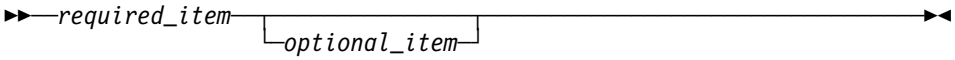
  The —►◄ symbol indicates the end of a statement.

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

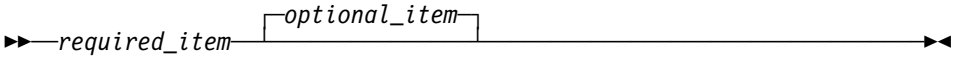
- Required items appear on the horizontal line (the main path).

  ►►—*required_item*—►◄

- Optional items appear below the main path.

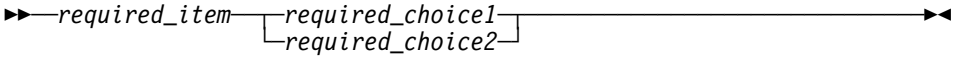


If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

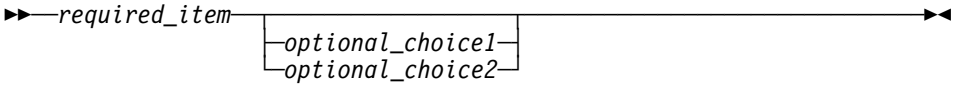


- If you can choose from two or more items, they appear vertically, in a stack.

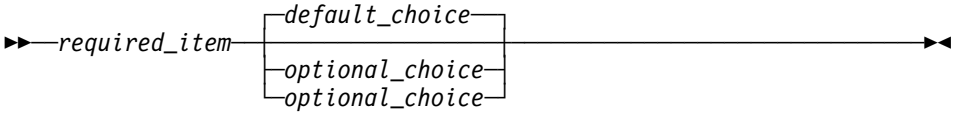
  If you *must* choose one of the items, one item of the stack appears on the main path.



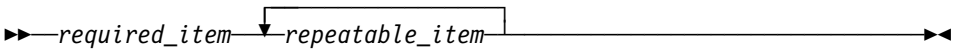
  If choosing one of the items is optional, the entire stack appears below the main path.



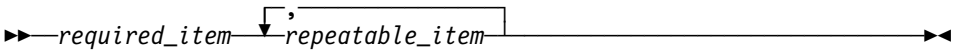
  If one of the items is the default, it appears above the main path and the remaining choices are shown below.



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



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



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

- 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.

Naming Conventions

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

This section describes naming conventions unique to commands and utilities. Characters are classified as *letters*, *digits*, or *special characters*.

- A *letter* is any one of the uppercase characters A through Z (plus 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 *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 *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 *Administration Guide* for more information about correlation IDs.

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.

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 table name can contain one or two parts, depending upon its qualification. 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 each of the parts.

table-space-name
A short identifier that designates a table space of an identified database. 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 for OS/390 Version 5. 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 *Release Guide*):

Installation: If you are involved with DB2 only to install the system, *Installation Guide* might be all you need.

If you will be using data sharing then you also need *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 *Application Programming and SQL Guide* and *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 some 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 *Application Programming and SQL Guide*, *SQL Reference*, and *Call Level Interface 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 Call Level Interface 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 *Application Programming and SQL Guide* and *Application Programming Guide and Reference for Java*. The material needed for writing applications that use DB2 Call Level Interface or ODBC to access DB2 servers is in *Call Level Interface Guide and Reference*.

For handling errors, see *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. *Administration Guide* divides those tasks among the following sections:

- Section 2 (Volume 1) of *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 *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 *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 *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 *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 *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:

- *SQL Reference*, which describes the SQL statements you use to create, alter, and drop objects and grant and revoke privileges

- *Utility Guide and Reference*, which explains how to run utilities
- *Command Reference*, which explains how to run commands

If you will be using data sharing, then you need *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 *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 *Diagnosis Guide and Reference* and *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. Technical professionals can access DB2 publications on the Web and follow links to other Web sites with more information about DB2 family and OS/390 solutions. Access DB2 on the Web with the following URL:

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

DB2 Publications

The DB2 publications are available in both hardcopy and softcopy 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 on the MVS, VM, OS/2, DOS, AIX and Windows platforms.

When you order DB2 Version 5, you are entitled to one copy of the following CD-ROM, which contains the DB2 licensed book for no additional charge:

DB2 Server for OS/390 Version 5 Licensed Online Book, LK2T-9075.

You can order multiple copies for an additional charge by specifying feature code 8207.

When you order DB2 Version 5, you are entitled to one copy of the following CD-ROM, which contains the DB2 and DATABASE 2 Performance Monitor online books for no additional charge:

DB2 Server for OS/390 Version 5 Online Library, SK2T-9092

You can order multiple copies for an additional charge through IBM's publication ordering service.

Periodic updates will be provided on the following collection kit available to licensees of DB2 Version 5:

IBM Online Library Transaction Processing and Data Collection, SK2T-0730

SK2T-9092 will be superseded by SK2T-0730 when updates to the online library are available.

In some countries,including the United States and Canada, you receive one copy of the collection kit at no additional charge when you order DB2 Version 5. You will automatically receive one copy of the collection kit each time it is updated, for no additional charge. To order multiple copies of SK2T-0730 for an additional charge, see “How to Order the DB2 Library.” In other countries, updates will be available in displayable softcopy format in the IBM Online Book Library Offering (5636–PUB), SK2T-0730 IBM Online Library Transaction Processing and Data Collection at a later date.

See your IBM representative for assistance in ordering the collection.

DB2 Server for OS/390 books are also available for an additional charge on the following collection kits, which contain online books for many IBM products:

IBM Online Library MVS Collection, SK2T-0710, in English

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 platforms

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 & SLSS option. Be prepared to give your customer number, the product number, and the feature code(s) or order numbers you want.

DB2 Classes

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. 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 diagrams below show the DB2 curriculum in the United States. *Enterprise Systems Training Solutions, GR28-5467* describes these courses. You can inquire about or enroll in them by calling 1-800-IBM-TEACh (1-800-426-8322).

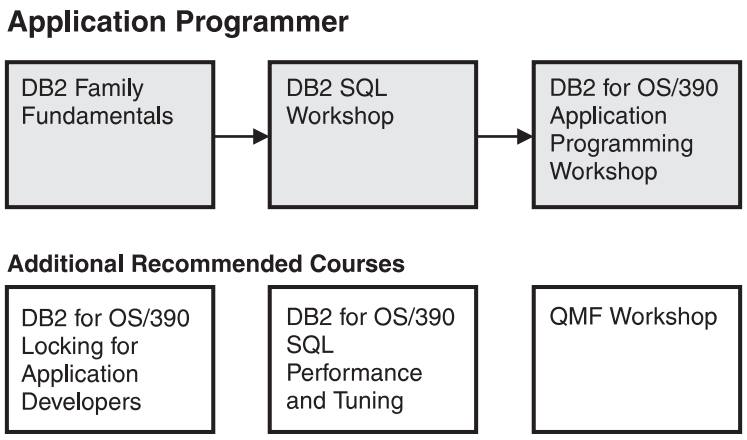


Figure 1. Application Programmer Curriculum

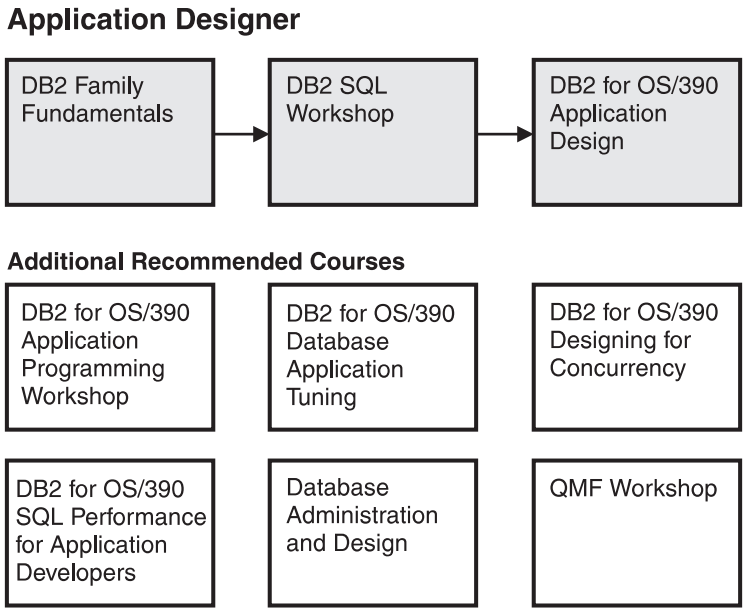


Figure 2. Application Designer Curriculum

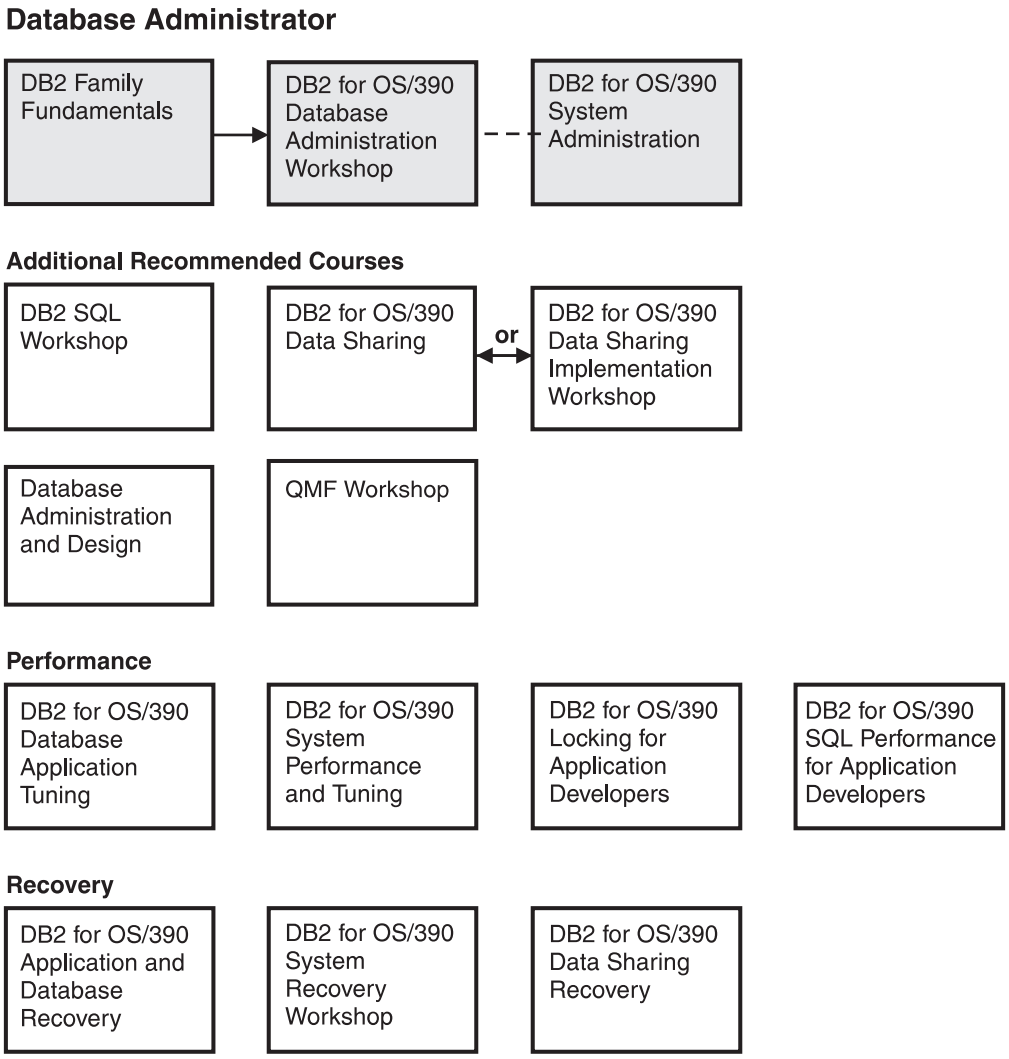


Figure 3. Database Administrator Curriculum

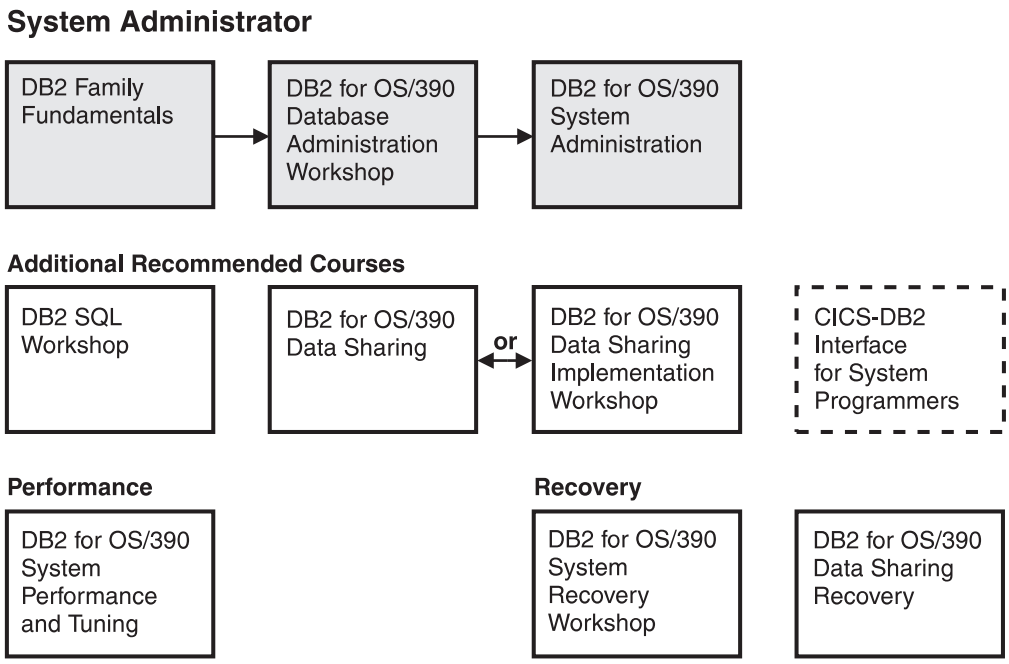


Figure 4. System Administrator Curriculum

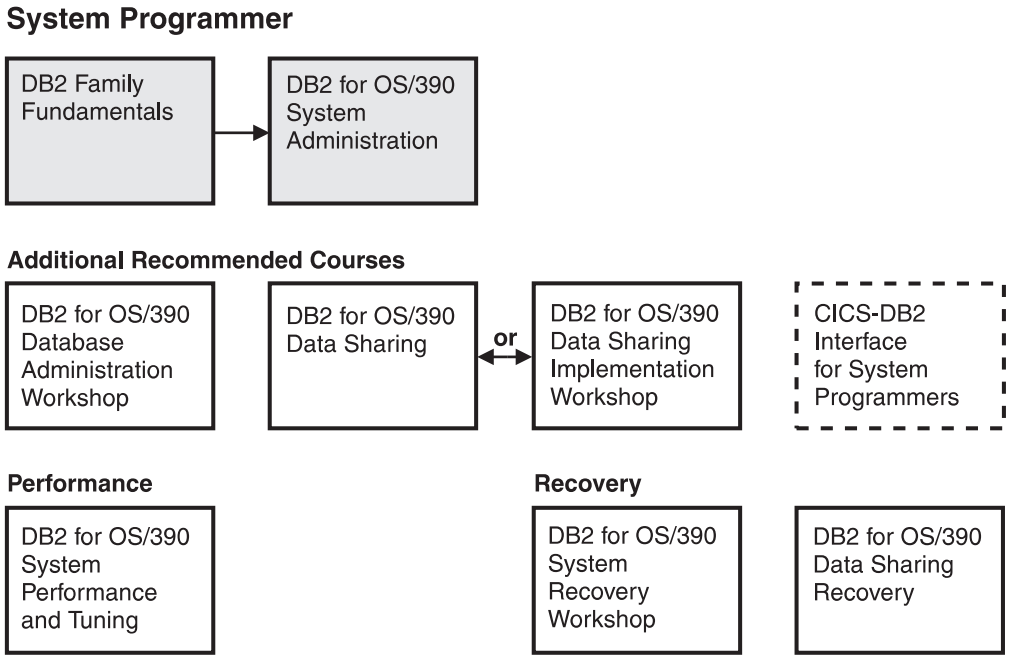


Figure 5. System Programmer Curriculum

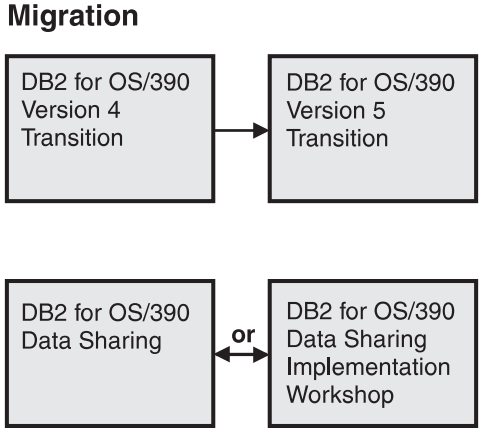


Figure 6. Migration Curriculum

Summary of Changes to DB2 for OS/390 Version 5

DB2 for OS/390 Version 5 delivers a database server solution for OS/390. Version 5 supports all functions available in DB2 for MVS/ESA Version 4 plus enhancements in the areas of performance, capacity, and availability, client/server and open systems, and user productivity.

If you are currently using DB2, **you can migrate only from a DB2 for MVS/ESA Version 4 subsystem**. This summary gives you an overview of the differences to be found between these versions.

Server Solution

OS/390 retains the classic strengths of the traditional MVS/ESA operating system, while offering a network-ready, integrated operational environment.

The following features work directly with DB2 for OS/390 applications to help you use the full potential of your DB2 subsystem:

- Net.Data for OS/390
- DB2 Installer
- DB2 Estimator for Windows
- DB2 Visual Explain
- Workstation-based Performance Analysis and Tuning
- DATABASE 2 Performance Monitor

Net.Data for OS/390

Net.Data provides support for Internet access to DB2 data through a Web server. Applications built with Net.Data make data stored in any DB2 server more accessible and useful. Net.Data Web applications provide continuous application availability, scalability, security, and high performance.

This no charge feature can be ordered with DB2 Version 5 or downloaded from Internet. The Net.Data URL is:

http://www.ibm.com/software/data/net.data/downloads.html

DB2 Installer

DB2 Installer offers the option to install DB2 on an OS/2 workstation. Now, you can use a friendly graphical interface to complete installation tasks easily with DB2 Installer.

This function is delivered on CD-ROM with DB2 Visual Explain.

DB2 Estimator for Windows

DB2 Estimator provides an easy-to-use capacity planning tool. You can estimate the sizes of tables and indexes, and the performance of SQL statements, groups of SQL statements (transactions), utility runs, and groups of transactions (capacity runs). From a simple table sizing to a detailed performance analysis of an entire DB2 application, DB2 Estimator saves time and lowers costs. You can investigate the impact of new or modified applications on your production system, *before* you implement them.

This no charge feature can be ordered with DB2 Version 5 or downloaded from the Internet. From the internet, use the IBM Software URL:

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

From here, you can access information about DB2 Estimator using the download function.

DB2 Visual Explain

DB2 Visual Explain lets you tune DB2 SQL statements on an OS/2 workstation. You can see DB2 EXPLAIN output in a friendly graphical interface and easily access, modify, and analyze applications with DB2 Visual Explain.

Workstation-based Performance Analysis and Tuning

The new workstation-based Performance Analysis and Tuning function simplifies system administration. You can access statistical data to help you analyze and improve system performance. This function works with the optional DB2 PM feature to provide full analysis and tuning functionality.

DATABASE 2 Performance Monitor (DB2 PM)

DB2 PM lets you monitor, analyze, and optimize the performance of DB2 Version 5 and its applications. An online monitor, for both host and workstation environments, provides an immediate "snap-shot" view of DB2 activities and allows for exception processing while the system is operational. The workstation-based online monitor can connect directly to the Visual Explain function of the DB2 base product.

DB2 PM also offers a history facility, a wide variety of customizable reports for in-depth performance analysis, and an EXPLAIN function to analyze and optimize SQL statements. For more information, see *DB2 PM for OS/390 General Information* .

This feature can be ordered with DB2 Version 5.

Performance

Sysplex Query Parallelism

The increased power of Sysplex query parallelism in DB2 for OS/390 Version 5 allows DB2 to go far beyond DB2 for MVS/ESA Version 4 capabilities; from the ability to split and process a single query within a DB2 subsystem to processing that same query across many different DB2 subsystems in a data sharing group.

The advances this release offers in scalable query processing let you process queries quickly while accommodating the potential growth of data sharing groups and the increasing complexity of queries.

Prepared Statement Caching

DB2 reduces the cost of duplicate prepares for the same dynamic SQL statement by saving them in a cache. Now, different application processes can share prepared statements and they are preserved past the commit point. This performance improvement offers the most benefit for:

- Client/server applications that frequently use dynamic SQL for repeated execution of SQL statements
- Relatively short dynamic SQL statements for which PREPARE cost accounts for most of the CPU expended

Reoptimization

When host variables, parameter markers, or special registers were used in previous releases, DB2 could not always determine the best access path because the values for these variables were unknown. Now, you can tell DB2 to reevaluate the access path at run time, after these values are known. As a result, queries can be processed more efficiently, and response time is improved.

Faster Transactions and Batch

- Caching of package authorization improves performance at run time for remote packages and applications that use pattern-matching characters in a package list.
- You can define a table space to use ***selective partition locking***, which can reduce locking costs for applications that do partition-at-a-time processing. It also can reduce locking costs for certain data sharing applications that rely on an affinity between members and data partitions.
- A new standalone utility lets you preformat active logs.
- With LOAD and REORG, you can preformat data sets up to the high allocated RBA, which can make processing for sequential inserts more predictable.

Faster Utilities

- LOAD and REORG jobs run faster and more efficiently with enhanced index key sorting that reduces CPU and elapsed time, and an inline copy feature that lets you make an image copy without a separate copy step.
- New REORG options let you select rows to discard during a REORG and, optionally, write the discarded records to a file.
- When you run the REBUILD, RECOVER, REORG, or LOAD utility on DB2-managed indexes or table spaces, a new option lets you logically reset and reuse the DB2-managed objects.

- RECOVER INDEX and LOAD run faster on large numbers of rows per page.
- Sampling support for RUNSTATS reduces the processing required to collect nonindexed column statistics.
- BSAM striping improves the I/O capability of DB2 utilities.

Other Performance Enhancements

- There are several significant performance enhancements to data sharing, including selective partition locking, the MAXROWS option, and several optimizations to reduce data sharing overhead.
- DB2 installations that run in the OS/390 Version 2 Release 6 environment can now have as many as (approximately) 25 000 open DB2 data sets at one time. The maximum number of open data sets in earlier releases of OS/390 is 10000.
- You can easily alter the length of variable-length character columns using the new ALTER COLUMN clause of the ALTER TABLE statement.
- SQL CASE expressions let you eliminate queries with multiple UNIONs and improve performance by using only one table scan.
- You can collect a new statistic on concatenated index keys to improve the performance of queries with correlated columns. The statistic lets DB2 estimate the number of rows that qualify for the query more accurately, and select access paths more efficiently.
- DB2 scans partitions more efficiently and allows scans during parallel processing.
- Query enhancements include the ability to:
  - Use indexes for joins on string columns that have different lengths
  - Use an index to access predicates with noncorrelated IN subqueries
- Noncolumn expressions in simple predicates are evaluated at stage 1 and can be indexable.

Increased Capacity

DB2 for OS/390 Version 5 introduces the concept of a *large* partitioned table space. Defining your table space as large allows a substantial capacity increase: to approximately one terabyte of data and up to 254 partitions. In addition to accommodating growth potential, large partitioned table spaces make database design more flexible, and can improve availability.

Improved Availability

Online REORG

DB2 for OS/390 Version 5 adds a major improvement to availability with *Online REORG*. Now, you can avoid the severe availability problems that occurred while offline reorganization of table spaces restricted access to read only during the unload phase and no access during reload phase of the REORG utility. Online REORG gives you full read and write access to your data through most phases of the process with only very brief periods of read only or no access.

Data Sharing Enhancements

- Version 5 provides continuous availability with group buffer pool duplexing. Prior releases of DB2 rely on DASD and the merged recovery logs to recover group buffer pool (GBP) data that is lost if a coupling facility fails. With group buffer pool duplexing, DB2 writes changed pages to both a *primary GBP* and a *secondary GBP.* Overlapped writes to the GBPs provide good performance and eliminate the writes to DASD.
- Group buffer pool rebuild makes coupling facility maintenance easier and improves access to the group buffer pool during connectivity losses.
- Automatic group buffer pool recovery accelerates GBP recovery time, eliminates operator intervention, and makes data available faster when GBPs are lost because of coupling facility failures.
- Improved restart performance for members of a data sharing group reduces the impact of retained locks by making data available faster when a group member fails.
- Changes to traces and DISPLAY GROUPBUFFERPOOL output improve monitoring.

Tracker site for disaster recovery

You can set up a tracker site that shadows the activity of a primary site, and eliminate the need to constantly ship image copies.

Client/Server and Open Systems

Native TCP/IP Network Support

DB2's support of TCP/IP networks allows DRDA clients to connect directly to DDF and eliminate the gateway machine. In addition, customers can now use asynchronous transfer mode (ATM) as the underlying communication protocol for both SNA and TCP/IP connections to DB2.

Stored Procedures

- Return multiple SQL result sets to local and remote clients in a single network operation.
- Receive calls from applications that use standard interfaces, such as Open Database Connectivity** (ODBC) and X/Open** Call Level Interface, to access data in DB2 for OS/390.
- Run in an enhanced environment. DB2 supports multiple stored procedures address spaces managed by the MVS Workload Manager (WLM). The WLM environment offers efficient program management and allows WLM-managed stored procedures to run as subprograms and use RACF security.
- Use individual MVS dispatching priorities to improve stored procedure scheduling.
- Access data sources outside DB2 with two-phase commit coordination.
- Use an automatic COMMIT feature on return to the caller that reduces network traffic and the length of time locks are held.
- Have the ability to invoke utilities, which means you can now invoke utilities from an application that uses the SQL CALL statement.

- Support IMS Open Database Access (ODBA). Now a DB2 stored procedure can directly connect to IMS DBCTL and access IMS data.

Dynamic Query and Network Performance

Improvements for DRDA Applications

- Reduced processing costs for block fetch operations
- DRDA support for OPTIMIZE FOR n ROWS on SELECT
- Faster dynamic SQL queries and reduced processing costs for VTAM network operations
- Reduced message traffic for dynamic SQL SELECT statements

Improved Application Portability

- DB2 for OS/390 Version 5 introduces the DB2 Call Level Interface (CLI) to MVS/ESA. Unlike applications that use embedded SQL to access DB2 data, applications that choose CLI are not tied to a precompiler, packages, or a plan.

  Workstation and desktop applications use standard interfaces, such as Open Database Connectivity (ODBC), to access relational data. Standard interfaces need one version of an application to access many data sources. Now, you can port UNIX workstation and PC desktop applications to DB2 for OS/390 and exploit the CLI (ODBC) capabilities without modification. In addition, applications can issue ODBC or CLI calls from within a stored procedure.
- You can now access DB2 for OS/390 databases in your Java applications. You can use DB2 Connect Java Database Connectivity (JDBC) for your dynamic SQL applications, or SQLJ for your static SQL applications.
- DB2 adds DRDA support for the DESCRIBE INPUT statement to improve performance for many ODBC applications.
- Now, you can write multithreaded DB2 CLI applications, and restrictions on connection switching no longer exist.
- DB2 now provides ASCII table support for clients and servers across platforms. This support reduces the cost of translation between EBCDIC and ASCII encoding schemes. ASCII table support also offers an alternative to writing field procedures that provide the ASCII sort sequence, which improves performance.

Improved Security

- DB2 for OS/390 supports Distributed Computing Environment (DCE) for authenticating remote DRDA clients. DCE offers the following benefits:
  - Network security: By providing an encrypted DCE ticket for authentication, remote clients do not need to send an MVS password in readable text.
  - Simplified security administration: End users do not need to maintain a valid password on MVS to access DB2; instead, they maintain their DCE password only.
- New descriptive error codes help you determine the cause of network security errors.
- You can change end user MVS passwords from DRDA clients.

User Productivity

Improved SQL Compatibility

DB2 conforms to the ANSI/ISO SQL entry level standard of 1992. Application programmers can take advantage of a more complete set of standard SQL to use across the DB2 family to write portable applications. New SQL function includes:

- More check options for view definitions.
- Foreign keys that reference UNIQUE keys as well as PRIMARY keys.
- An extension to GRANT that lets the REFERENCES privilege apply to a list of columns.
- A new delete rule, NO ACTION, that you can use to define referential constraints for self-referencing tables.
- SQL CASE expressions provide the capability to create conditional logic wherever an expression is allowed.
- SQL temporary tables allow application programs to easily create and use temporary tables that store results of SQL transactions without logging or recovery.

New Access Choice

A new attachment facility, the Recoverable Resource Manager Services attachment facility, improves access in a client/server environment. It coordinates two-phase commit processing between DB2 and other participating resource managers in any MVS application environment. Other key features include the ability for multiple users to run in a single address space, thread reuse, and moving threads between MVS tasks.

Image Copy Enhancements

The COPY, LOAD, and REORG utilities provide:

- Features of the COPY utility that help you quickly determine what type of image copy to take, when to take it, and let DB2 automatically take it for you.
- Inline copy in LOAD and REORG that lets you create an image copy while improving data availability.

Improved Integration of C++ and IBM COBOL for MVS & VM Support

It is easier for application programmers to use object-oriented programming techniques in their DB2 applications. DB2 for OS/390 Version 5 adds COBOL and C++ languages as options on installation panels, DB2I panels, the DSNH command, and DCLGEN.

Other Usability Enhancements

- To prevent long running units of work and to help avoid unnecessary work during the recovery phase of restart, DB2 issues new warning messages at an interval of your choice.
- A new special register for decimal precision provides better granualarity, so that applications that need different values for decimal precision can run in the same DB2 subsystem.
- Trace records for IFCID 0022 now include most information in the PLAN_TABLE.

- An increase from 127 to 255 rows on a page improves table space processing and eliminates the need for compression.
- Install SYSOPR can recover objects using the START DATABASE command.
- A filtering capability for DISPLAY BUFFERPOOL limits statistics information to a specified set of page sets.
- You can enter comments within the SYSIN input stream for DB2 utilities.

Summary of Changes to This Book

There are general changes affecting invocation and syntax of all utilities for Version 5 of DB2 for OS/390:

The syntax of all the utility commands will accept the entry of comments. See "Control Statement Coding Rules" on page 2-7

There are changes to the following online utilities.

"Chapter 2-6. COPY" on page 2-57
"Chapter 2-8. LOAD" on page 2-87
"Chapter 2-14. REORG" on page 2-197
"Chapter 2-17. RUNSTATS" on page 2-277

There are changes to the following stand-alone utilities:

"Chapter 3-2. DSNJLOGF (Preformat Active Log)" on page 3-7
"Chapter 3-6. DSN1COMP" on page 3-43
"Chapter 3-7. DSN1COPY" on page 3-49
"Chapter 3-9. DSN1PRNT" on page 3-85

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 below.

Description of Online Utilities

DB2 online utilities run as standard MVS batch jobs, 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 2-7 for information about the ways to invoke these utilities.

Description of Stand-alone Utilities

The stand-alone utilities execute as batch jobs independent of DB2. They can be invoked only by means of MVS JCL. Refer to the chapters on the individual utilities in “Chapter 3-1. Invoking Stand-Alone Utilities” on page 3-5 for information about the ways to invoke 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.

There are three types of identifiers: primary authorization IDs, secondary authorization IDs, and SQL IDs.

- Generally it is the primary authorization ID that 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 *Administration Guide*.

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

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

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 the four different methods available 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 four available methods for invoking the online utilities. Each method is described separately in one of the following sections:

- "Using the DB2 Utilities Panel in DB2I" on page 2-11
- "Using the DSNU CLIST in TSO" on page 2-14
- "Using the Supplied JCL Procedure (DSNUPROC)" on page 2-20
- "Creating the JCL Yourself" on page 2-23
- # "Invoking utilities as a stored procedure (DSNUTILS)" on page X-7

For the least involvement with JCL, use either DB2I or the DSNU CLIST, 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 closely with JCL, or supply your own JCL, choose one of the last two methods listed above.

Creating Utility Control Statements

Utility control statements define the function 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 fixed-length 80-character records, columns 73 through 80 are ignored.
- The records are concatenated before being parsed; hence, 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	CHECK DATA	CHECK INDEX
COPY	DIAGNOSE	LOAD
MERGECOPY	MODIFY	QUIESCE
RECOVER TABLESPACE	RECOVER INDEX	REORG
REPAIR	REPORT	RUNSTATS
STOSPACE		

  At least one blank character must follow the name.

- Other syntactical constructs in the utility control statement describe options; they can be separated from each other by an arbitrary number of blanks.
- The SYSIN stream can contain multiple utility control statements.

The online utility name determines which options can follow it. More than one utility control statement can be specified in the SYSIN stream.

Options are typically described by an *option keyword*, followed by a *parameter*. The parameter value can be a keyword. Parameters' values are sometimes enclosed in parentheses, but are not generally required to be. The syntax diagrams for utility control statements included in Chapter 2-1. Invoking DB2 Online Utilities show parentheses where they are required.

You can enter comments within the SYSIN data 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
- Comments can be started wherever a space is valid, except within a delimiter token.
- Comments are terminated by the end of the line.

```
// 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 above, parameter values can be enclosed in parentheses but are not always required to be. The description of the temporary work file can be written as either

`WORKDDN SYSUT1` or `WORKDDN (SYSUT1)`

Data Sets Used by Online Utilities

A SYSIN DD statement is needed in every online utility job to describe an input data set; some utilities also need other data sets. Table 1 on page 2-9 lists the name of each DD statement that could be needed, the online utilities that require it, and the purpose of the corresponding data sets. If an alternate DD name is allowed, you give it as a parameter in a utility option. The table also lists the option keywords used. DCB attributes that are specified 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 Concurrent Copy using filter.	FILTERDDN
DATAWK*nn*	REORG	Work data set for sorting data. *nn* is a 2-digit number; you can use several data sets. To estimate the size of the data set needed, see "Creating JCL Statements for Required Data Sets" on page 2-199.	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	LOAD[1,3,7], REORG[1,7], CHECK DATA[2,7]	Holds sorted keys (sort output) to allow SORT phase to be restartable; for CHECK DATA, holds sorted keys (sort output).	WORKDDN
SORTWK*nn* [4]	CHECK DATA, CHECK INDEX, LOAD, REORG, RECOVER INDEX	Work data set for sorting indexes. *nn* is a 2-digit number; you can use several data sets. To estimate the size of the data set needed, see page 2-90.	—
SYSCOPY	COPY, MERGECOPY, LOAD[5], REORG [5]	Output data set for copies.	COPYDDN RECOVERYDDN
SYSDISC	LOAD	Discarded records (optional).	DISCARDDN
SYSERR	CHECK DATA[2], LOAD	Contains information about errors encountered during processing.	ERRDDN
SYSIN	All	Input data set for utility statements.	—
SYSMAP	LOAD[3]	Contains information about what input records violated a constraint.	MAPDDN
SYSPRINT	All	Messages and printed output (usually SYSOUT).	—
SYSREC	LOAD[2], REORG[6]	LOAD input data set; unloaded records for REORG.	INDDN UNLDDN
SYSUT1	CHECK DATA[7], CHECK INDEX[2], LOAD[1,3,7], REORG[1,7], RECOVER INDEX, MERGECOPY	Temporary work data set that holds sorted keys for input to the SORT phase; for MERGECOPY, holds intermediate merged output.	WORKDDN
UTPRINT	CHECK DATA, CHECK INDEX, LOAD, REORG, RECOVER INDEX	Messages from DFSORT (usually, SYSOUT or DUMMY).	

Table 1 (Page 2 of 2). Data Sets Used by Online Utilities

DD Name	Used By	Purpose	Option Keyword

Notes::

1 Required for tables with indexes, except not required for REORG when SORTKEYS is used.
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 NOSYSREC or SHRLEVEL CHANGE is specified.
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 user-specified values are supplied for LRECL or RECFM, they are ignored. If a user-specified block size exists, 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 user-specified number of buffers (BUFNO), with a maximum of 99 buffers. The default number of buffers is 20.[1] 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 any DCB parameters are omitted, the utilities choose default values.

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

To prevent unauthorized access to data sets (for example, image copies), you can protect the data sets with passwords or by using 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.

Concatenating Data Sets

DB2 utilities support the concatenation of unlike input data sets. Therefore, the data sets in a concatenation list can have differing block sizes, logical record lengths, and record formats. In order to concatenate variable and fixed blocked data sets, the logical record length must be 8 bytes smaller than the block size.

Output data sets cannot be concatenated.

---

1 This might require an increased region size for jobs that ran in DB2 Version 3. You can use the SIZE= option of DSNUPROC to increase the size. See “Using the Supplied JCL Procedure (DSNUPROC)” on page 2-20 for information about using DSNUPROC.

Controlling Data Set Disposition

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

- Retain the image copy data sets until they are no longer needed for recovery.
- Retain the unload data sets if UNLOAD(PAUSE) or UNLOAD(ONLY) has been specified for the REORG utility.
- Retain the discard data set until the contents are no longer needed for subsequent loads.

Because you could 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 are discarded 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.

Using the DB2 Utilities Panel in DB2I

This method of invoking DB2 online utilities requires the least knowledge of JCL.

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 must submit your job directly from the editor. You can use the procedure outlined in the following steps:

1. Create the utility control statement for the online utility you intend to invoke, 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 DSN8D51A.DSN8S51D with a SHRLEVEL value of CHANGE:

   ```
   COPY TABLESPACE DSN8D51A.DSN8S51D
     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. This panel is shown in Figure 7. Items you must enter are highlighted.

```
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, CHECK DATA, COPY, DIAGNOSE, LOAD,
                             MERGE, MODIFY, QUIESCE, RECOVER INDEX,
                             RECOVER TABLESPACE, REORG INDEX,
                             REORG TABLESPACE, REPORT, REPAIR,
                             RUNSTATS, STOSPACE.)

  4 CONTROL CARDS DATA SET    ===> UTIL

 To RESTART a utility, specify starting point, otherwise enter 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 7. 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, enter SUBMIT on the editor command line.
5. Field 2 must be a unique identifier for your utility job. When the panel appears, it contains 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. In this example, enter 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, enter 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 select 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 selected.

```
DSNEUP02                          DATA SET NAMES
 ===>

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

   Enter data set name for LOAD:
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 ==>


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

Figure 8. DATA SET NAMES Panel

1. Fill in field 1 if you are running LOAD or REORG. If you are running LOAD, you must indicate the data set name that contains records to be loaded. If you are running REORG you must indicate 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 with discard processing; you must indicate 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. Press ENTER.

If you need help while using the DB2 utilities panel or the data set names panel, press the HELP PF key. There are HELP panels to explain the parameters on the DB2 Utilities panel, and HELP panels that show the syntax and some sample utility control statements for each online utility.

Using the DSNU CLIST in TSO

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

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

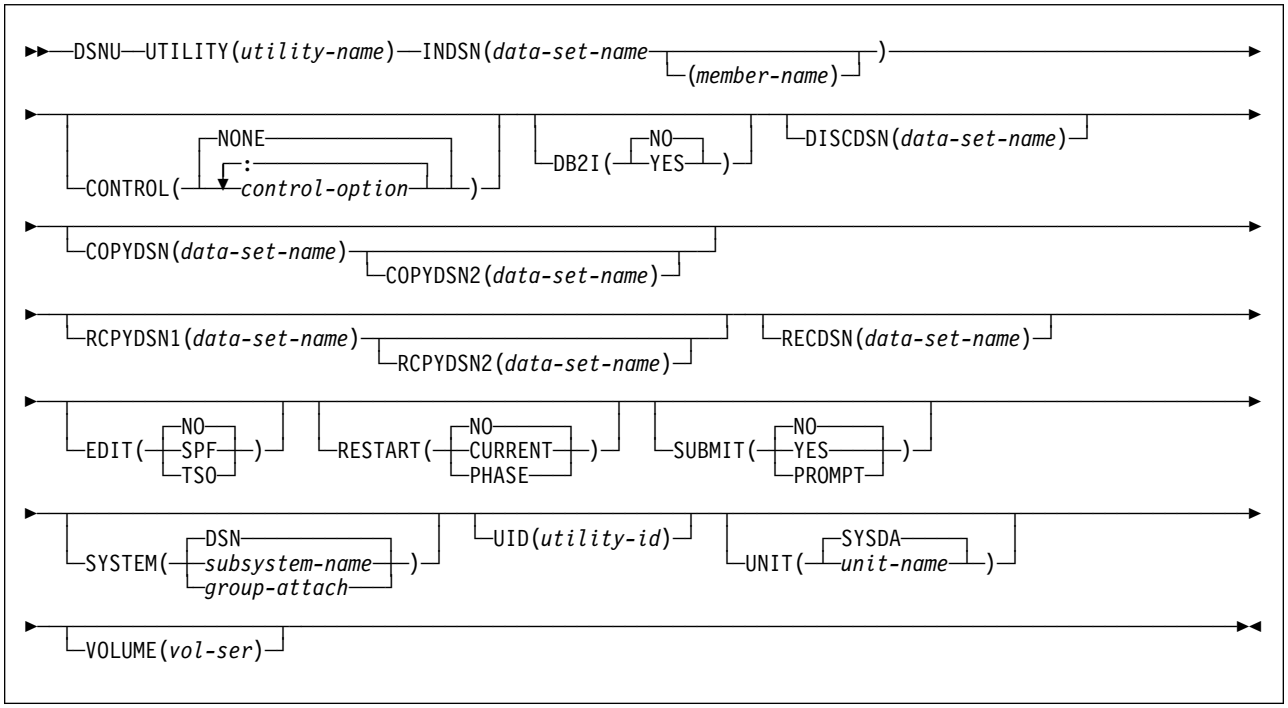
To use the DSNU CLIST:

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. Note that this file must not include double-byte character set (DBCS) data.
2. Ensure that the DB2 CLIST library is allocated to ddname SYSPROC.
3. Invoke the command procedure, using the syntax that follows.
4. Edit the generated JCL to alter or add DD statements as needed.

   This last step is optional. “Editing the Generated JCL” on page 2-19 explains the steps you can take.

The DSNU CLIST can be invoked from the TSO command processor or the DB2I utilities panel.

DSNU CLIST Syntax



Option Descriptions

The parentheses shown in the following descriptions are required. If you make syntax errors or omit parameter values, TSO prompts you for 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 invoke. Select the name from the following list:

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

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, invokes the utility. If you invoke another job with the same utility name, the first job is deleted. See Table 2 on page 2-18 for a list of the online utilities and the control file name associated with each utility.

INDSN(*data-set-name* **(***member-name***))**
Tells what data set contains the utility statements and control statements. Note that the data set you specify must not contain double-byte character set data.

(*data-set-name***)**
Is the name of the data set.

The **default** is that the command procedure prompts you for the data set name.

(*member-name***)**
Is the member name, and must be given if the data set is partitioned.

CONTROL(*control-option***: ...)**
Tells whether to trace the CLIST 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, code 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 that DSNU CLIST is called from.

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

The **default** is **DB2I**(**NO**).

(YES)
Indicates that DSNU CLIST is being called from the DB2I environment and therefore possible JOB card information can be used. Only the utility panels should invoke the CLIST with DB2I(YES).

DISCDSN(*data-set-name***)**
Is the cataloged data set name used by LOAD as a discard data set, to hold records not loaded.

COPYDSN(*data-set-name***)**
Is the name of the cataloged data set required as a target (output) data set. If you do not supply this information, the CLIST prompts you for it. It is required for COPY and MERGECOPY, and for these utilities the CLIST prompts you for this information if you do not supply 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***)**
Is 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.

RCPYDSN1(*data-set-name***)**
Is 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.

RCPYDSN2(*data-set-name***)**
Is 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.

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

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

(NO)
Does not invoke an editor.

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

(SPF)
Invokes the ISPF editor.

(TSO)
Invokes the TSO editor.

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

(NO)
Indicates the utility is a new job, not a restart. There must not be any other utility job step with the same utility identifier (UID).

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. The current stopped phase can be determined by the DISPLAY UTILITY command.

SUBMIT
Tells whether to submit the generated JCL for processing.

(NO)
Does not submit the JCL 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 tell whether to submit the JCL data set for batch processing. You cannot use PROMPT when the CLIST itself 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 given in the following table:

Table 2. Control File Names

Utility	*control-file-name*	**Utility**	*control-file-name*
CHECK	DSNUCHE	CHECK DATA	DSNUCHD
COPY	DSNUCOP	DIAGNOSE	DSNUDIA
LOAD	DSNULOA	MERGECOPY	DSNUMER
MODIFY	DSNUMOD	QUIESCE	DSNUQUI
RECOVER TABLESPACE	DSNURCT	RECOVER INDEX	DSNURCI
REORG INDEX	DSNURGI	REORG TABLESPACE	DSNURGT
REPAIR	DSNUREP	REPORT	DSNURPT
RUNSTATS	DSNURUN	STOSPACE	DSNUSTO

UNIT(*unit-name***)**
Gives 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***)**
Gives 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 Output

DSNU builds a one-step job stream. The JCL consists of a job statement, an EXEC statement that invokes the DB2 utility processor and the required DD statements, including the SYSIN DD * job stream, as shown in Figure 9. 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.DSN8D51A.JAN1,DISP=(MOD,CATLG,CATLG),
//          UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN     DD    *
     COPY TABLESPACE DSN8D51A.DSN8S51D
          FULL NO
          SHRLEVEL CHANGE

/*
```

Figure 9. DSNUCOP.CNTL. This is the JCL before editing.

The following list describes the required statements:

Statement	Description
JOB	The CLIST uses any JOB statements that you had saved when using DB2I. If there are no existing JOB statements, DB2 produces a skeleton JOB statement that you can modify after the JCL is complete. The jobname is DSNU, followed by the first three letters of the name of the utility you are using.
EXEC	The CLIST 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 DSNUPROC.

The CLIST builds the necessary JCL DD statements. Those statements vary according to the utility being invoked. Data sets that could be required are listed under “Data Sets Used by Online Utilities” on page 2-8.

SYSPRINT DD SYSOUT=A
Utility messages are sent to SYSPRINT. The generated JCL defines SYSPRINT as SYSOUT=A. You can use the TSO OUTPUT command to control the disposition of the SYSPRINT data set; for example, you can have it sent to your terminal. For further information, see *TSO/E Command Reference.*

UTPRINT DD SYSOUT=A
If any utility requires a sort, it invokes 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 is not changed, and you can reuse it when the DSNU procedure has completed.

Editing the Generated JCL

You can request to edit the data set before you submit 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, you must change the ddname in the JCL generated by DSNU. For example, in the REORG TABLESPACE utility the default 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.

At the end of the edit, you can either save changes to the data set (by issuing SAVE), or instruct the editor to ignore all changes.

The SUBMIT parameter tells whether the data set is to be submitted as a background job. The temporary data set holding the JCL is reused. If you want to submit more than one job that invokes the same utility, you must rename the JCL data sets and submit them separately.

Examples

Example 1: This CLIST generates a data set called *authorization-id*.DSNURGT.CNTL that contains JCL to invoke the DSNUPROC procedure. This procedure invokes the REORG TABLESPACE utility. MYREOR.DATA will be 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 as a batch job and will not be modified by this CLIST until a new request is made to invoke the REORG TABLESPACE utility.

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

Example 2: The following is an example of how to invoke the CLIST for the COPY utility.

```
%DSNU
  UTILITY (COPY)
  INDSN ('MYCOPY(STATEMNT)')
  COPYDSN ('MYCOPIES.DSN8D51A.JAN1')
  EDIT (TSO)
  SUBMIT (YES)
  UID (TEMP)
  RESTART (NO)
```

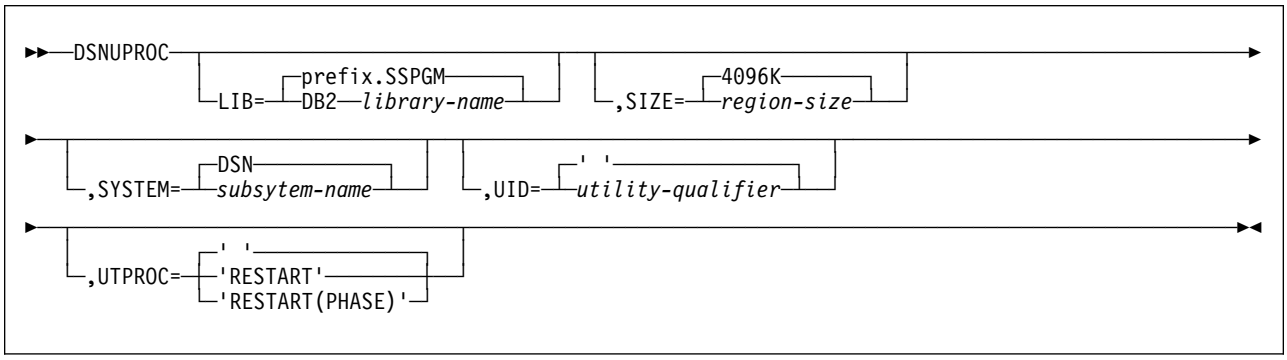
Using the Supplied JCL Procedure (DSNUPROC)

Another way to initiate a DB2 online utility is by using the supplied JCL procedure, DSNUPROC, shown in Figure 10 on page 2-22. This procedure uses the parameters that you supply to build an appropriate EXEC statement to invoke an online utility.

To invoke the DSNUPROC procedure, you must write and submit JCL like that built by the DSNU CLIST, and shown in Figure 9 on page 2-18. In your JCL, the EXEC statement invokes the DSNUPROC procedure. The parameters you can supply to that procedure, and their syntax, are explained in the following section.

DSNUPROC Syntax

Use the syntax in the following figure to write the EXEC DSNUPROC statement in the JCL.



Option Descriptions

The following list describes all the parameters. For the example, you need to use only one, 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 **4096K**.

SYSTEM=
| Specifies the DB2 subsystem or group attach name.

| The **default** is **DSN**.

UID=
Specifies the unique identifier for your utility job. The name can use up to 16 characters. If the name contains special characters, enclose it between apostrophes: for example, 'PETERS.JOB'.

The **default** is an empty string.

UTPROC=
Controls restart processing.

The **default** is an empty string, used when you are not restarting a stopped job.

To restart the utility, use:

'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 whatever other data sets are needed by your job. See “Data Sets Used by Online Utilities” on page 2-8 for a description of data sets that could be needed.

Figure 10 on page 2-22 is the DSNUPROC procedure invoked by the JCL example in Figure 9 on page 2-18.

Sample DSNUPROC Listing

```
//DSNUPROC PROC LIB='prefix.SSPGM',
//         SYSTEM=DSN,
//         SIZE=4096K,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 5                                       *
//*                                                                   *
//* 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.           *
//*           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.    *
```

Figure 10 (Part 1 of 2). Sample Listing of Supplied JCL Procedure (DSNUPROC)

```
//*           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 10 (Part 2 of 2). Sample Listing of Supplied JCL Procedure (DSNUPROC)

Creating the JCL Yourself

DB2 online utilities execute as standard OS/VS jobs. In order 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 use an EXEC statement and a set of DD statements. The EXEC statement is described in the following section. For a description of the DD statements you could need, see “Data Sets Used by Online Utilities” on page 2-8.

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]'
```

In this statement, 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
Is the unique identifier for your utility job.

utproc
Is the value of the UTPROC parameter in DSNUPROC. Use it only when restarting the utility job. Use:

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

In our example, 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.

The following sections describe how to monitor utility status and how to terminate an online utility.

Monitoring Utilities With the DISPLAY UTILITY Command

The information under this heading, up to "Running Utilities Concurrently" on page 2-26 is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page ix.

You can learn the current status of online utilities by issuing the DB2 DISPLAY UTILITY command. DB2 returns a message indicating the member name (**A**), utility name (**B**), identifier (**C**), phase (**D**), and status (**E**). The message also indicates the number of pages or records processed by the utility (**F**)[2]. It may also include additional information for an executing utility, such as log phase estimates (not shown).

An example of the DISPLAY UTILITY command's output is shown below.

```
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
```

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 2-27).

[2] 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.

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; see "Phases and Data Flow" in the descriptions of each utility. Output from -DISPLAY UTILITY shows the currently executing phase.

Determining Why a Utility Failed to Complete

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

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

- **Problem:** DB2 terminates the utility job step and any later 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 there is no duplicate utility running.

- **Problem:** The particular utility function is not executed by DB2, but prior utility functions are executed.

  Solution: Submit a new utility step to execute the function.

- **Problem:** The utility function is placed in the stopped state by DB2.

  Solution: Restart the utility job step at either the last commit point or the beginning of the phase, using the same utility identifier. Alternately, terminate the job step (using -TERM UTILITY (uid)) and resubmit it.

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 this chapter feature a 'Compatibility and Concurrency' section. The section gives the following information, usually in tabular form:

- For each target object the utility acts on, the claim classes that the utility must claim or drain and any restrictive state that the utility sets on the target.
- For other online utilities, whether or not they are compatible with the given utility on the same target object. If two actions are compatible on a target object, they can be 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 *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 the group attach name is not used, 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. There are several MVS installation-specific ways to make sure this happens. These include:

- For JES2 multi-access spool (MAS) systems,

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

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

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

The *MVS/ESA JCL Reference* describes the above JCL statements. 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 can restart that DB2 on another MVS system and terminate the utility from that system.

A utility may be restarted on any member, but that member must be able to access all required data sets. The same utility ID (UID) must be used to restart the utility. That UID is unique within a data sharing group. However, if a DB2 fails, the DB2 must be restarted on either the same or another MVS before the utility can be restarted.

Running Online Utilities on a Shared Read-Only Database

All online utilities that you can normally run on a database can be run against an owner's database. However, you need to take special steps before running the following utilities: COPY, LOAD, RECOVER, REORG, REPAIR (DELETE and REPLACE).

For steps to run the above utilities, in addition to further information regarding shared databases, see Appendix F (Volume 2) of *Administration Guide*

Terminating an Online Utility With the TERM UTILITY Command

The information under this heading, up to “Restarting an Online Utility” on page 2-28 is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page ix.

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, the terminated utility cannot be restarted. In addition, it is possible that the objects on which the utility was operating can 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 could have to terminate it in order to make the data available to other applications. To terminate, 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 could choose to put -TERM UTILITY in a conditionally executed job step; for example, if you never want to restart certain utility jobs. Figure 11 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 11. Example of conditionally executed -TERM UTILITY

Restarting an Online Utility

If a utility finishes abnormally, it is often possible to restart it. With restart, 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 phase does not take any checkpoints or has not reached the first checkpoint, current restart is equivalent to phase restart.

Updating the JCL For Restarting a Utility

To restart a DB2 online utility, update the original JCL 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 shown in Figure 8 on page 2-13, except for field 5.
  3. Change field 5 to CURRENT or PHASE, depending upon the method of restart desired.
  4. Press ENTER.
- *Using the DSNU CLIST*. Restart the utility by invoking the DSNU CLIST as described in "Using the DSNU CLIST in TSO" on page 2-14, but change the value of the RESTART parameter, using either RESTART(CURRENT) or RESTART(PHASE).
- *Creating your own JCL*. If you create your own JCL, you must 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 an Out of Space Condition

There are special considerations when restarting a utility at the last commit point after receiving an out of space condition on the output data set (for example, ABENDx37 on SYSUT1).

The following DB2 Utilities and output datasets might be affected because they allow restart from the last commit point.

Utility	Phase	Commit Point Restartable Output Data Set
COPY	copy	SYSCOPY (other COPYDDN) RECOVERYDDN
MERGECOPY	mergecopy	SYSUT1 (other WORKDDN) SYSCOPY (other COPYDDN) RECOVERYDDN
LOAD	reload	SYSUT1, SYSERR, SYSMAP
REORG	unload (see note) reload	SYSREC SYSUT1, SYSERR, SYSMAP

Note: If SORTDATA is not specified.

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

1. Copy the output data set to a temporary data set, using the same DCB parameters. Use MVS utilities that do not reblock the data set during the copy operation (i.e. DFDSS ADRDSUU, DFSORT ICEGENER).
2. Delete or rename the output data set. Redefine the data set with additional space, using the same VOLSER (if the data set is not cataloged), the same DSNAME, and the same DCB parameters. You should know present 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 (i.e. DFDSS ADRDSUU, DFSORT ICEGENER).

   In steps 1 and 3 above, do not use MVS utilities that reblock the dataset during the copy operation (i.e., IEBGENER, ISPF 3.3).

Other Restart Hints

- The VOLSER (if the data set is not cataloged) and the data set name are used to keep track of utility restart information. They must be the same as the original utility run.
- Do not change the utility function that is currently stopped and the DB2 objects upon which it is operating. However, you can change other parameters relating to the stopped step and later utility steps.
- Do not specify MVS automatic step restart.
- When restarting a utility, do not specify VOL=SER=. Let DB2 determine the data sets from the system catalog.
- If a utility is restarted in the UTILINIT phase, it is re-executed from the beginning of the phase.
- If a utility abends or system failure occurs while the utility is in the UTILTERM phase, it must be restarted 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 has two functions, UPDATE and CONVERT. UPDATE updates the catalog; it should be run only when migrating to a new release of DB2. CONVERT converts catalog and directory indexes from type 1 to type 2 indexes. You can also use CATMAINT CONVERT to convert type 2 indexes to type 1. It converts only IBM-defined catalog indexes.

Syntax Diagram: For a diagram of CATMAINT syntax and a description of available options, see “Syntax Diagram” on page 2-34.

Output: Output from CATMAINT CONVERT consists of converted indexes and a message for each index indicating that the conversion was successful. Output for CATMAINT UPDATE consists of the updated catalog.

Authorization Required: One of the following authorities is needed to run CATMAINT CONVERT:

- DBADM on DSNDB06
- SYSCTRL
- SYSADM
- Installation SYSADM
- Installation SYSOPR

The authorization required for CATMAINT UPDATE is installation SYSADM.

Instructions for Running CATMAINT

In order to run CATMAINT, you must:

1. Prepare the necessary data sets, as described in “Creating JCL Statements for Required Data Sets” on page 2-32.
2. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for CATMAINT, see “Sample JCL and Control Statements” on page 2-35.)
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 2-33. (For a complete description of the syntax and options for CATMAINT, see “Syntax and Options of the Control Statement” on page 2-34.)
4. Check “Concurrency and Compatibility” on page 2-34 if you want to run other jobs concurrently on the same target objects.
5. Plan for restart if the CATMAINT job doesn't complete, as described in “Terminating or Restarting CATMAINT” on page 2-34.

Invoking CATMAINT

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for an explanation of ways to invoke DB2 utilities.

Phases and Data Flow

The CATMAINT UPDATE utility operates in these phases:

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

The CATMAINT CONVERT utility operates in these phases:

Phase	Description
UTILINIT	Initialization and setup
CONVERT	Set all the indexes to recovery pending status and change the DBD to the new type
CNVBUILD	Rebuild the index
UTILTERM	Cleanup

Creating JCL Statements for Required Data Sets

Table 3 describes the data sets used by CATMAINT. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 3. Data Sets Used by CATMAINT

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:

Indexes Objects to be converted. You designate on the control statement whether you wish to convert catalog or directory indexes.

Creating the Control Statement

See “Syntax and Options of the Control Statement” on page 2-34 for CATMAINT syntax and option descriptions. See “Sample JCL and Control Statements” on page 2-35 for examples of CATMAINT usage.

Before Running CATMAINT

Calculating the Size of the Work File Data Base: The work file data base is used for CATMAINT sorting. To calculate the size of the work file data base, see Section 2 (Volume 1) of *Installation Guide*.

Calculating the Size of Type 2 Indexes: Type 2 indexes are larger than type 1 indexes. You may need to increase the space allocation parameters for your indexes in order to convert them. See Section 2 (Volume 1) of *Installation Guide* for instructions on calculating the size of type 2 indexes.

Instructions for Specific Tasks

For the tasks described, specify the options and values shown with your utility control statement. The following tasks are described here:

"Updating the Catalog for a New Release"
"Converting Type 1 Indexes to Type 2"

Updating the Catalog for a New Release

When you migrate to a new release of DB2, you must update the prior release's catalog to the new version. The DSNTIJTC job, described in Section 2 (Volume 1) of *Installation Guide*, runs CATMAINT with the UPDATE option to update the catalog. CATMAINT UPDATE should only be run when migrating to a new release.
| See "Migrating the DB2 Subsystem" in Section 2 of *Installation Guide* for the steps
| necessary to migrate to a new release.

Converting Type 1 Indexes to Type 2

Run CATMAINT CONVERT TO TYPE 2 to convert your catalog or directory
indexes to type 2. Ensure that DSNDB07 is large enough to handle the sort when
using CATMAINT. For better concurrency, you should convert all indexes of your catalog and directory tables to type 2.

Reviewing CATMAINT CONVERT Output

CATMAINT CONVERT returns a message for each index. A message DSNU768 indicates that the conversion was successful, while a message DSNU769 indicates that the conversion failed. If one index fails, CATMAINT CONVERT continues to convert the other indexes and terminates with a return code 8.

If a conversion fails, the index is put in recovery pending status. If it is a critical index, such as an index used to check authorization, you may need to use an ID with installation SYSADM or installation SYSOPR authority to run the recovery or rerun CATMAINT CONVERT.

If CATMAINT CONVERT terminates with a return code 8, correct the problem and then either:

1. Run RECOVER INDEX on the indexes that failed, or
2. Rerun CATMAINT CONVERT to rebuild all the indexes.

After Running CATMAINT CONVERT

When you convert an index from type 2 to type 1, plans and packages dependent on the index are invalidated and must be rebound. When you convert an index from type 1 to type 2, plans and packages dependent on the index are not invalidated and do not need to be rebound.

Displaying CATMAINT Status: The DISPLAY UTILITY command does not display CATMAINT status.

Terminating or Restarting CATMAINT

You can terminate CATMAINT UPDATE with the TERM UTILITY command, but it leaves the indexes in "recovery pending" status. See "Resetting the Recovery Pending Status" on page 2-104 for information on resetting this status.

If CATMAINT CONVERT fails, review the output and execute `DISPLAY DB(dbname)SPACENAM(*)RESTRICT`. This command shows which indexes are in recovery pending status. Terminate CATMAINT CONVERT. Either submit the CATMAINT CONVERT job again or submit a RECOVER job for the indexes that are in recovery pending status.

CATMAINT cannot be restarted.

For more guidance in restarting online utilities, see "Restarting an Online Utility" on page 2-28.

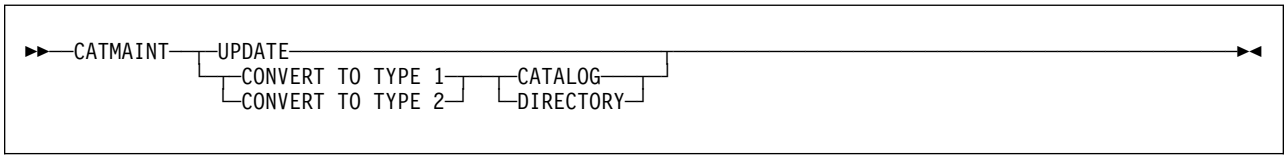
Concurrency and Compatibility

Catalog and Directory Index Unavailability: The catalog or directory indexes are unavailable while CATMAINT is running. This can cause other jobs to time out with message DSNT376I or get a resource not available message DSNT501I.

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



Option Descriptions

UPDATE
Indicates that the intent is to update the catalog. This option should be run only when migrating to a new release of DB2.

CONVERT TO TYPE1
Converts the indexes to type 1.

CONVERT TO TYPE 2
Converts the indexes to type 2.

CATALOG
Converts the catalog indexes to the specified type.

DIRECTORY
Converts the directory indexes to the specified type.

Sample JCL and Control Statements

Example

Example: Converting Catalog Indexes to Type 2: This example converts the catalog indexes to type 2 indexes:

```
//UTILSAMP EXEC PGM=DSNUTILB,REGION=1024K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*
//*********************************************************************
//*  DATA SETS USED BY THE UTILITY                                    *
//*********************************************************************
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
    CATMAINT CONVERT TO TYPE 2 CATALOG
```

Chapter 2-4. CHECK DATA

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

CHECK DATA should be run after a conditional restart or a point-in-time recovery on all table spaces where parent and dependent tables might not be synchronized.

Syntax Diagram: For diagram of CHECK DATA syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 2-44.

Output: CHECK DATA optionally deletes rows in violation of 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.

Upon 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 data base
- DBADM, DBCTRL, or DBMAINT authority for the data base
- SYSCTRL or SYSADM authority

An ID with installation SYSOPR authority can also execute CHECK DATA, but only on table spaces *besides* SYSDBASE in the DSNDB06 data base. (CHECK DATA cannot be run on the SYSDBASE table space, or on any object in the DSNDB01 data base except SYSUTILX.)

If the DELETE option is specified, then the privilege set must include DELETE privilege on the tables being checked. If the FOR EXCEPTION option is specified, then the privilege set must include INSERT privilege on any exception table used.

Instructions for Running CHECK DATA

In order to run CHECK DATA, you must:

1. Read “Before Running CHECK DATA” on page 2-39 in this chapter.
2. Prepare the necessary data sets, as described in “Creating JCL Statements for Required Data Sets” on page 2-38.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for CHECK DATA, see “Sample JCL and Control Statements” on page 2-47.)
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 2-41. (For a complete description of the syntax and options for CHECK DATA, see “Syntax Diagram” on page 2-44.)
5. Check the compatibility table in “Concurrency and Compatibility” on page 2-43 if you want to run other jobs concurrently on the same target objects.

6. Plan for restart if the CHECK DATA job doesn't complete, as described in “Terminating or Restarting” on page 2-43.

Invoking CHECK DATA

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for a description of ways to invoke DB2 utilities.

Phases and Data Flow

One of the following phases can be identified if the job terminates.

The phases for CHECK DATA are:

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

Creating JCL Statements for Required Data Sets

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 (Page 1 of 2). Data Sets Used by CHECK DATA

Data Set	Description	Required?
SYSIN	Input data set containing the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
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. To find the approximate size in bytes of the work data sets, see page 2-39.	Yes
Error data set	Output data set that collects information about errors encountered during the CHECKDAT phase for referential constraint violations or the SCANTAB phase for check constraint violations. Its DD name is specified with the ERRDDN parameter of the utility control statement. The default DD name is SYSERR.	Yes
UTPRINT	Contains messages from DFSORT (usually, SYSOUT or DUMMY).	No

Table 4 (Page 2 of 2). Data Sets Used by CHECK DATA

Data Set	Description	Required?
SORTLIB	Contains the DFSORT load module. If you omit this, and sorting is needed, SYS1.SORTLIB is dynamically allocated.	No

The following objects are named in the utility control statement and do not require DD cards 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, you must use the PART option in the control statement.)

Exception tables
For each table in a table space being checked, you can specify the name of an exception table in the utility control statement. Any row in violation of a referential constraint is copied to the associated exception table. See page 2-39 for more information.

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

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

1. Add 13 to the length of the longest foreign key.
2. Multiply the sum by the number of keys checked.

Create the ERRDDN data set so that it is large enough to accommodate 1 error entry (length=60 bytes) per defect detected by CHECK DATA.

Creating the Control Statement

See “Syntax Diagram” on page 2-44 for CHECK DATA syntax and option descriptions. See “Sample JCL and Control Statements” on page 2-47 for examples of CHECK DATA usage.

Before Running CHECK DATA

Running CHECK INDEX First: 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.

Creating Exception Tables: An exception table is a user-created table that duplicates the definition of a dependent table. The dependent table is the table being checked with the CHECK DATA utility. It consists of at least *n* columns, where *n* is the number of columns of the dependent table for which it is used. The CHECK DATA utility copies the deleted rows from the dependent table to the exception table. Table 5 describes the contents of an exception table.

Table 5. Exception Tables

Column	Description	Required	Data Type and Length	NULL Attribute
1 to n	Corresponds to columns in the table being checked and are used to hold data from rows in the table being checked 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) for LARGE table spaces	Anything
n+2	Starting time of the CHECK utility	No	TIMESTAMP	Anything
≥ *n+2*	Any additional columns are not used by the CHECK utility	No	Anything	Anything

If you delete rows with CHECK DATA, you must have exception tables for all tables in the table spaces named and for all their descendents, because you will delete all descendents of any row you delete.

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 could cause errors when CHECK DATA inserts rows in them.
- You can create a new exception table prior to running 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.
- Exception table column names can be given any name.
- Any change to the structure of the dependent table (such as a column dropped or added) is not automatically reflected in the exception table. You must make that change to the exception table yourself.

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 using these SQL statements:

```
CREATE TABLE EPROJACT
   LIKE DSN8510.PROJACT
   IN DATABASE DSN8D51A;

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 DSN8510.PROJACT as well as the privileges usually required to create a table.

Table EPROJACT has the same structure as table DSN8510.PROJACT, but with 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 something else. But the description, CHAR(4), is required.
- The final timestamp column is also useful. If the timestamp column is defined, a row identifier (RID) column must exist and precede this column. You might define a permanent exception table for each table subject to referential or table check constraints. You can define it once and use it as you wish to hold invalid rows detected by CHECK DATA. The TIME column allows you to identify rows added by the most recent run.

Eventually, you make corrections to the data in the exception tables, perhaps with an SQL UPDATE, and transfer the corrections to the original tables with statements like this one:

```
INSERT INTO DSN8510.PROJACT
   SELECT PROJNO, ACTNO, ACSTAFF, ACSTDATE, ACENDATE
     FROM EPROJACT
     WHERE TIME > CURRENT TIMESTAMP - 1 DAY;
```

End of General-use Programming Interface

Instructions for Specific Tasks

For the tasks described, specify the options and values shown with your utility control statement. The following tasks are described here:

Specifying CHECK DATA Scope

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

Checking 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

Correcting Constraint Violations

You can automatically delete rows causing violations of referential or table check constraint by specifying CHECK DATA with DELETE YES. However, you should be aware of the following:

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

Deletion could make the true error harder to detect and correct, or could cause valid rows to be deleted. Because of this, you should run CHECK DATA with DELETE NO to detect the violations, before you attempt to correct the errors. If required, DELETE YES can be used after you have analyzed the output and understand the errors.

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

Resetting Check Pending Status

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

The check pending status can be removed by:

- Using the option DELETE YES to remove all rows in violation of referential or table check constraints.

- Using the option DELETE NO if no tables contain rows that violate referential or table check constraints.

Terminating or Restarting

For instructions on restarting a utility job, see “Restarting an Online Utility” on page 2-28.

Stopping CHECK DATA

When you terminate CHECK DATA, table spaces remain in the check pending status in which 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 the DELETE YES option was specified.

Restarting CHECK DATA

You can restart a CHECK DATA utility job, although it starts over again 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. However, if a type 1 nonpartitioned index exists on a partitioned table space, utilities operating on different partitions of a table space can be incompatible because of contention on the nonpartitioned index.

Claims and Drains: Table 6 shows which claim classes CHECK DATA claims and drains and any restrictive state the utility sets on the target object.

Table 6 (Page 1 of 2). CHECK DATA. Use of claims and drains; restrictive states set.

TARGET	CHECK DATA DELETE NO	CHECK DATA DELETE YES	CHECK DATA PART DELETE NO	CHECK DATA PART DELETE YES
Table space/Partition	DW/UTRO	DA/UTUT	DW/UTRO	DA/UTUT
Partitioned Index/Partition	DW/UTRO	DA/UTUT	DW/UTRO	DA/UTUT
Nonpartitioned index type 1	DW/UTRO	DA/UTUT	DW/UTRO	DA/UTUT
Nonpartitioned index type 2	DW/UTRO	DA/UTUT		DR
Logical partition of type 2 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

Table 6 (Page 2 of 2). CHECK DATA. Use of claims and drains; restrictive states set.

TARGET	CHECK DATA DELETE NO	CHECK DATA DELETE YES	CHECK DATA PART DELETE NO	CHECK DATA PART DELETE YES
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

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.

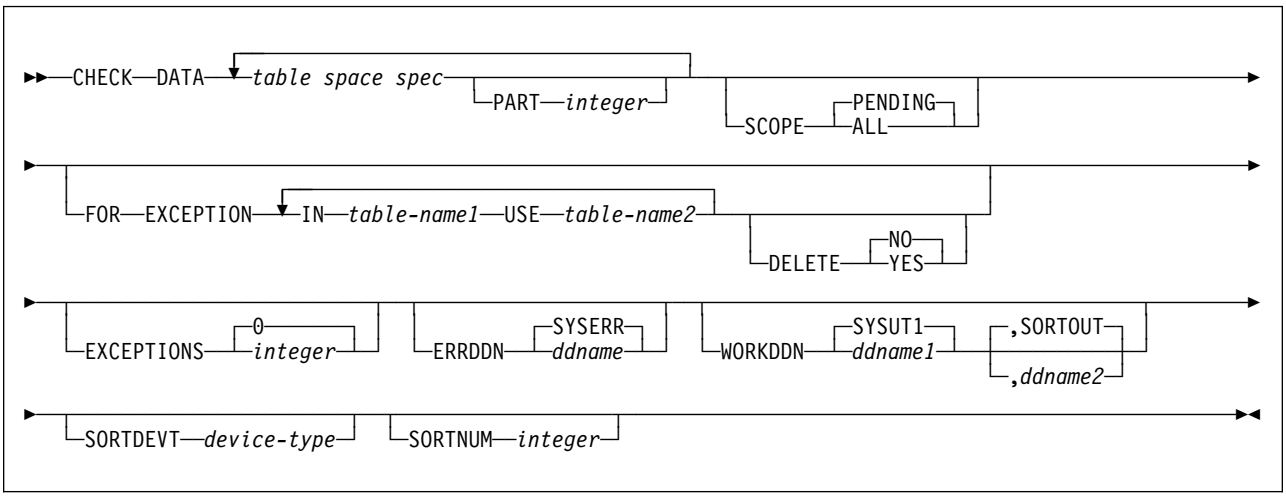
To run on DSNDB01.SYSUTILX, CHECK DATA must be the only utility in the job step and the only utility running in the DB2 subsystem.

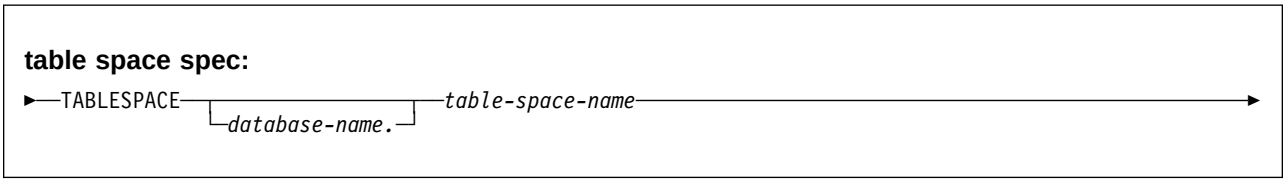
Syntax and Options of the Control Statement

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 1-3.





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 2-7.

DATA
Indicates that you are checking referential and table check constraints.

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 which are to be checked.

PENDING
Indicates that only those rows that need to be checked in table spaces, partitions, or tables that are in “check pending” status are to be checked.

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**.

ALL
Indicates that all dependent tables in the specified table spaces are to be checked.

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. Output consists of deleted rows from both dependent and descendent tables.

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 needs 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.

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.

Sample JCL and Control Statements

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

```
//UTILSAMP JOB USER=SYSADM,PASSWORD=SYSADM,NOTIFY=SYSADM,
//         REGION=4M,TIME=1440,MSGCLASS=A,CLASS=A
//UTILSAMP EXEC PGM=DSNUTILB,REGION=4096K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*
//*********************************************************************
//*  DATA SETS USED BY THE UTILITY                                    *
//*********************************************************************
//SORTOUT  DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTWK01 DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSERR   DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSPRINT DD  SYSOUT=*
//UTPRINT  DD  SYSOUT=*
//SYSIN    DD  *

 CHECK DATA TABLESPACE DSN8D51A.DSN8S51D
            TABLESPACE DSN8D51A.DSN8S51E
      FOR EXCEPTION IN DSN8510.DEPT        USE DSN8510.EDEPT
                    IN DSN8510.EMP         USE DSN8510.EEMP
                    IN DSN8510.PROJ        USE DSN8510.EPROJ
                    IN DSN8510.PROJACT     USE DSN8510.EPROJACT
                    IN DSN8510.EMPPROJACT  USE DSN8510.EEPA
      DELETE YES
//*
```

Example 2: Control Statement for Deleting Error Rows: Check for and delete all constraint violations in table spaces DSN8D51A.DSN8S51D and DSN8D51A.DSN8S51E.

```
CHECK DATA TABLESPACE DSN8D51A.DSN8S51D
 TABLESPACE DSN8D51A.DSN8S51E
 FOR EXCEPTION IN DSN8510.DEPT USE DSN8510.EDEPT
               IN DSN8510.EMP USE DSN8510.EEMP
               IN DSN8510.PROJ USE DSN8510.EPROJ
               IN DSN8510.PROJECT USE DSN8510.EPROJECT
               IN DSN8510.EMPPROJECT USE DSN8510.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.

Syntax Diagram: For diagram of CHECK INDEX syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 2-53.

Output: CHECK INDEX generates several messages that show whether the indexes are consistent with the data. See Section 3 of *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 does issue an error message if there are 2 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 data base
- DBADM, DBCTRL, or DBMAINT authority for the data base
- 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 data bases.

Instructions for Running CHECK INDEX

In order to run CHECK INDEX, you must:

1. Prepare the necessary data sets, as described in “Phases and Data Flow” on page 2-50.
2. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for CHECK INDEX, see “Sample JCL and Control Statements” on page 2-55.)
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 2-51. (For a complete description of the syntax and options for CHECK INDEX, see “Syntax and Options of the Control Statement” on page 2-53.)
4. Check the compatibility table in “Concurrency and Compatibility” on page 2-52 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 2-52.

Invoking CHECK INDEX

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for an explanation of ways to invoke DB2 utilities.

Phases and Data Flow

CHECK INDEX operates in these phases:

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.

Creating JCL Statements for Required Data Sets

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 2-50.	Yes
UTPRINT	Contains messages from DFSORT (usually, SYSOUT or DUMMY).	No
SORTLIB	Contains the DFSORT load module. If you omit this, and sorting is needed, SYS1.SORTLIB is dynamically allocated.	No

The following objects are named in the utility control statement and do not require DD cards 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 needed 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. Multiply that sum by the largest key length plus 13.

Another way to estimate 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 2-53 for CHECK INDEX syntax and option descriptions. See “Sample JCL and Control Statements” on page 2-55 for examples of CHECK INDEX usage.

Instructions for Specific Tasks

For the tasks described, specify the options and values shown with your utility control statement. The following tasks are described here:

“Checking a Single Logical Partition”

Checking a Single Logical Partition

You can run CHECK INDEX on a single logical partition of a nonpartitioned type 2 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 inconsistencies are detected, you should analyze the output to determine the problem and then correct the inconsistency. If the data is correct and the index is in error, RECOVER INDEX can be used to re-establish consistency.

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 2-28.

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 type 1 nonpartitioned index exists on a partitioned table space, utilities operating on different partitions of a table space can be incompatible because of contention on the nonpartitioned index.

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. CHECK INDEX. Use of claims and drains; restrictive states set.

Target	CHECK INDEX	CHECK INDEX PART
Table space or partition	DW/UTRO	DW/UTRO
Partitioned index or partition	DW/UTRO	DW/UTRO
Nonpartitioned type 1 index	DW/UTRO	(1)
Nonpartitioned type 2 index	DW/UTRO	
Logical partition of type 2 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
- (1) CHECK INDEX PART is not allowed on a nonpartitioned type 1 index

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

Compatibility: Table 9 on page 2-53 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 shown.

Table 9. CHECK INDEX Compatibility

Action	CHECK INDEX
CHECK DATA	No
CHECK INDEX	Yes
COPY	Yes
DIAGNOSE	Yes
LOAD	No
MERGECOPY	Yes
MODIFY	Yes
QUIESCE	Yes
RECOVER	No
REORG INDEX	No
REORG UNLOAD CONTINUE or PAUSE	No
REORG UNLOAD ONLY	Yes
REPAIR DUMP or VERIFY	Yes
REPAIR DELETE or REPLACE	No
REPORT	Yes
RUNSTATS	Yes
STOSPACE	Yes

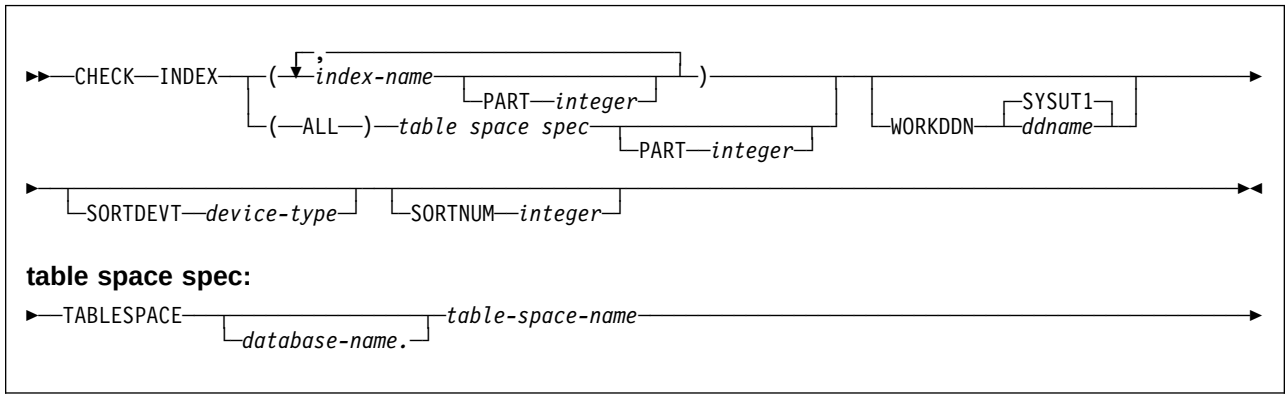
To run on SYSIBM.DSNLUX01 or SYSIBM.DSNLUX02, CHECK INDEX must be the only utility within the same job step.

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 1-3.



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 2-7.

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 partitioned index or a logical partition of a nonpartitioned type 2 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.

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 needs 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.

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.

Sample JCL and Control Statements

Examples

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

```
//UTILSAMP JOB USER=SYSADM,PASSWORD=SYSADM,NOTIFY=SYSADM,
//         REGION=4M,TIME=1440,MSGCLASS=A,CLASS=A
//*
//*        UTILITY SAMPLE JOB
//*
//UTILSAMP EXEC PGM=DSNUTILB,REGION=1024K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*********************************************************************
//*  DATA SETS USED BY THE UTILITY                                    *
//*********************************************************************
//SORTWK01 DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSPRINT DD  SYSOUT=*
//UTPRINT  DD  SYSOUT=*
//SYSIN    DD  *

 CHECK INDEX (ALL) TABLESPACE DSN8D51A.DSN8S51E

//*
```

Example 2: Check One Index: Check the project-number index (DSN8510.XPROJ1) on the sample project table.

```
CHECK INDEX (DSN8510.XPROJ1)
  SORTDEVT SYSDA
```

Example 3: Check More Than One Index: Check the indexes DSN8510.XEMPRAC1 and DSN8510.XEMPRAC2 on the employee to project activity sample table.

```
CHECK INDEX NAME (DSN8510.XEMPRAC1, DSN8510.XEMPRAC2)
```

Example 4: Check All Indexes on a Table Space: Check all indexes on the employee-table table space (DSN8S51E).

```
CHECK INDEX (ALL) TABLESPACE DSN8S51E
  SORTDEVT 3380
```

Chapter 2-6. COPY

The COPY online utility creates up to four image copies of a table space or a data set within a table space. There are two types of image copies:

1. A *full image copy* is a copy of all pages in a table space, partition, or data set.
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 TABLESPACE utility when recovering a table space to the most recent time or to a previous time.

Syntax Diagram: For diagram of COPY syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 2-71.

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 TABLESPACE utility. It is your installation's responsibility to ensure that these data sets are available if the RECOVER TABLESPACE utility requests them.
- If you specify the CHANGELIMIT option, a report on the change status of the table space or spaces.

The copy pending status is off if the copy was a full image copy.

Related information: See Section 4 (Volume 1) of *Administration Guide* for uses of COPY in the context of planning for data base recovery.

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

- IMAGCOPY privilege for the data base
- DBADM, DBCTRL, or DBMAINT authority for the data base
- 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 data base.

The batch user ID that invokes COPY with the CONCURRENT option must provide the necessary authority to invoke the DFDSS DUMP command.

Instructions for Running COPY

In order to run COPY, you must:

1. Read “Before Running COPY” on page 2-60 in this chapter.
2. Prepare the necessary data sets, as described in “Creating JCL Statements for Required Data Sets” on page 2-58.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for COPY, see “Sample JCL and Control Statements” on page 2-76.)

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 2-60. (For a complete description of the syntax and options for COPY, see “Syntax and Options of the Control Statement” on page 2-71.)
5. Check the compatibility table in “Concurrency and Compatibility” on page 2-69 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” on page 2-68.

Invoking COPY

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for an explanation of ways to invoke DB2 utilities.

Phases and Data Flow

The COPY utility operates in these phases:

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

Creating JCL Statements for Required Data Sets

Table 10 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 10. 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]
Copies	From 1 to 4 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 1 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
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, 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 table spaces, individual table space partitions, or table space linear pieces 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 could specify BUFNO=30. See also “Data Sets Used by Online Utilities” on page 2-8 for information about using BUFNO.

Cataloging Image Copies: To catalog your image copy data sets, use the DISP=(,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.

Before Running COPY

Checking Table Space Status: You cannot copy a table space that is in the check pending or recovery pending status. See “Resetting Recovery Pending Status” on page 2-186 for information about resetting these statuses.

Resetting Copy Pending Status: If the table space is in copy pending status, you can reset the status only by taking a full image copy of the entire table space or of
all its partitions. When you make an image copy of a pa rtition, 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.

Creating the Control Statement

See “Syntax Diagram” on page 2-71 for COPY syntax and option descriptions. See “Sample JCL and Control Statements” on page 2-76 for examples of COPY usage.

Instructions for Specific Tasks

For the tasks described, specify the options and values shown with your utility control statement. The following tasks are described here:

Making Full Image Copies

You can make a full image copy of a table space, a partition, or a data set within a table space. The following statement makes a full image copy of the DSN8S51E table space in data base DSN8D51A:

```
COPY TABLESPACE DSN8D51A.DSN8S51E
```

The COPY utility writes pages from the table space to the output data sets. The JCL for the utility job must include DD statements for the data sets. If the table space 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 table space 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.

| If you use create an inline copy during LOAD or REORG, you do not need to
| execute a separate COPY job. 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 a ny later 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 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 the parallel copy of other objects; therefore, it is most efficient to defer copying them until other copy jobs have completed. However, if you must copy other objects in parallel, 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 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 imbedded 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 DSN8D51A.DSN8S51E
  FULL NO
  SHRLEVEL CHANGE
```

Performance Advantage: An incremental image copy does not require a complete scan of the table space. 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.

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 a table space or data set. 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 in parallel in 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 data base 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 DSN8S51E in database DSN8D51A, 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 DSN8D51A.DSN8S51E
  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 copies under any of these conditions, COPY terminates with a return code of 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.

Maintaining Copy Consistency: If the table space is in copy pending status, or after a REORG procedure in which an inline copy was not created, make full image copies for both the local and recovery sites. This helps to insure correct recovery for both. 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 state 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 state at the current site, and you make copies from there for the other site only, the copy pending state will still be on when you bring up the system at that other site.

Performing Parallel Image Copy

If a table space is partitioned, you can copy several or all of the partitions independently in separate parallel 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 parallel jobs. To do so, run parallel COPY jobs (one job for each data set) and use SHRLEVEL CHANGE on each.

Using More than One COPY Statement

You can use more than one control statement for COPY in one DB2 utility job step. Doing that is useful for copying a complete set of referentially related table spaces after running QUIESCE. 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 using RESTART CURRENT 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 the table space you specify is segmented, 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 later run the RECOVER TABLESPACE 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 can specify a list of table spaces to copy.
- You must use the SHRLEVEL REFERENCE option for table spaces with a 32 KB 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 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 invoke the CONCURRENT option from the DB2I Utilities panel or from the DSNU TSO CLIST.

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.

Requirements for Using DFSMS Concurrent Copy: To use COPY to take DFSMS concurrent copies, you must have the following hardware and software:

- MVS Version 4 Release 3.
- DFSMS/MVS Version 1 Release 1.
- 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.

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.

Specifying a Percent Limit of Changed Pages: You can specify one or two percent values for CHANGELIMIT. If you specify only one value, COPY CHANGELIMIT:

- Creates an incremental image copy if the percentage of changed pages is greater than 0 and less than the value you specify
- Creates a full image copy if the percentage of changed pages is greater than or equal to the value you specify
- Does not create an image copy if no pages have changed.

If you specify two values, 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

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. 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 a list of table spaces, the object with the highest percentage of changed pages determines the return code and the recommended action against the entire list.

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, or if you have recently run REORG or LOAD, 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.

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 needed; see “Specifying Conditional Image Copies” on page 2-65.

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 on which mixed specific and non-specific volume IDs were defined with CREATE STOGROUP or ALTER STOGROUP.

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 needs. 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 2-66 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.

Terminating or Restarting

For instructions on restarting a utility job, see “Restarting an Online Utility” on page 2-28.

Warning Against TERM UTILITY

We do *not* recommend stopping a COPY step with the TERM UTILITY command.

If you issue TERM UTILITY while COPY is in the active or stopped state, a "T" # record is inserted in SYSIBM.SYSCOPY. When you run COPY, it does not allow an # incremental image copy if the "T" record exists. To reset the status, you must # make a full image copy. **Use RESTART CURRENT instead**, becaus e it:

- Is valid for full image copies and incremental copies
- Is valid for a single job step with several COPY statements
- Requires a minimum of re-processing
- Keeps the DB2 catalog and the integrated catalog facility catalog in agreement

For RESTART CURRENT, the same image copy data set is used. Therefore, specify DISP=(MOD,CATLG) on your DD statements.

Implications of DISP on the DD Statement

If you terminate a COPY job that uses the parameter DISP=(MOD,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, copy the failed copy output to the new data set before restarting.

Restarting a Full Image Copy

A full image copy can be restarted from either:

- The current utility commit point
- The beginning of a phase

Restarting an Incremental Image Copy

If -TERM UTILITY is *not* used, you can restart the job from the last commit point.

Specify RESTART CURRENT when restarting an incremental copy job, rather than restarting from the beginning of the phase. The copy job continues where it left off by positioning to the commit point position of the output data set.

Restarting COPY After an Out Of Space Condition

See “Restarting After an Out of Space Condition” on page 2-29 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. However, if a type 1 nonpartitioned index exists on a partitioned table space, utilities operating on different partitions of a table space can be incompatible because of contention on the nonpartitioned index.

Claims and Drains

Table 11 shows which claim classes COPY claims and drains and any restrictive state the utility sets on the target object.

Table 11. COPY. Use of claims and drains; restrictive states set.

Target	SHRLEVEL REFERENCE	SHRLEVEL CHANGE
Table space 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

* 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 12 on page 2-70 shows 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 12. COPY Compatibility

Action	SHRLEVEL REFERENCE	SHRLEVEL CHANGE
CHECK DATA	No	No
CHECK INDEX	Yes	Yes
COPY	No	No
DIAGNOSE	Yes	Yes
LOAD	No	No
MERGECOPY	No	No
MODIFY RECOVERY	No	No
MODIFY STATISTICS	Yes	Yes
QUIESCE	Yes	No
RECOVER INDEX	Yes	Yes
RECOVER TABLESPACE	No	No
REORG INDEX	Yes	Yes
REORG UNLOAD CONTINUE or PAUSE	No	No
REORG UNLOAD ONLY	Yes	Yes
REPAIR LOCATE by KEY, RID, or PAGE DUMP or VERIFY	Yes	Yes
REPAIR LOCATE by KEY or RID DELETE or REPLACE	No	No
REPAIR LOCATE TABLESPACE PAGE REPLACE	No	No
REPAIR LOCATE INDEX PAGE REPLACE	Yes	No
REPORT	Yes	Yes
RUNSTATS	Yes	Yes
STOSPACE	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.

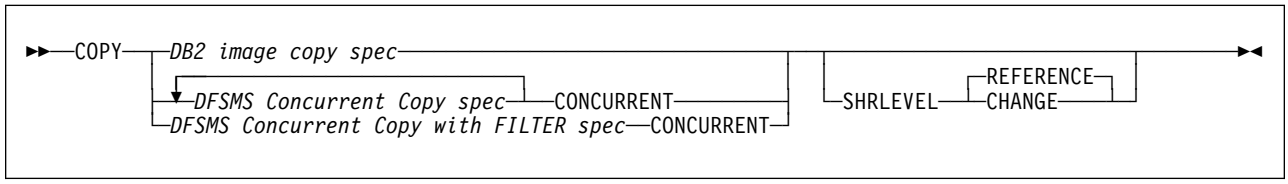
| COPY on SYSUTILX is an “exclusive” job; such a job can interrupt another job
| between job steps, possibly causing the interrupted job to time out.

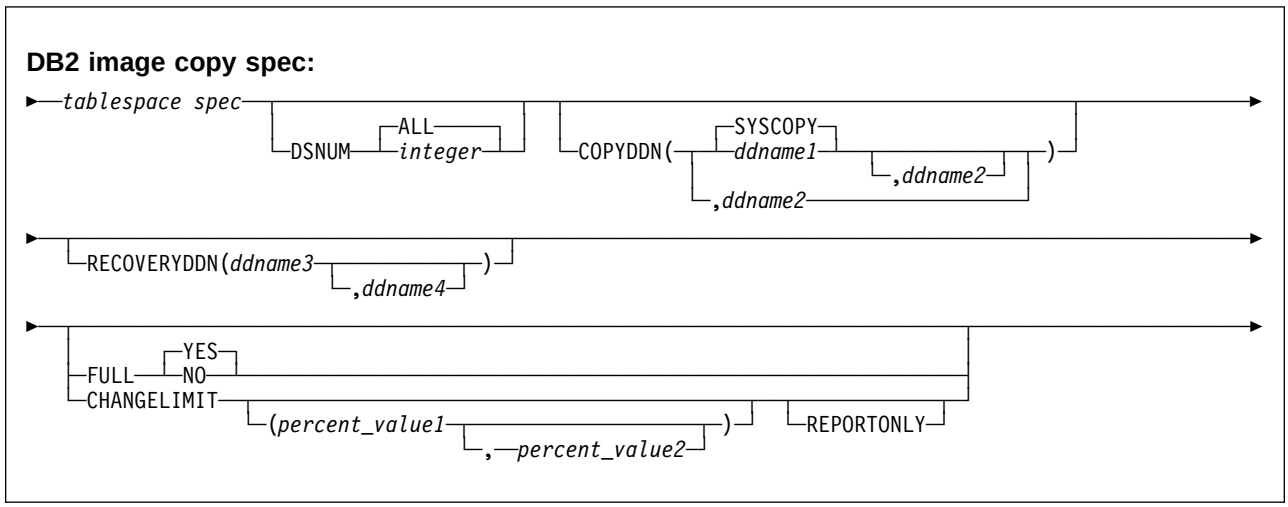
Syntax and Options of the Control Statement

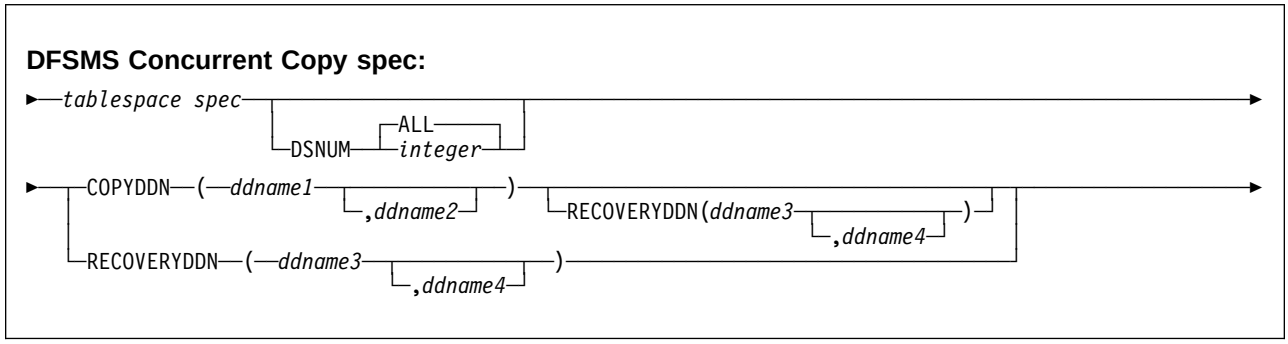
The utility control statement defines the function the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name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

Syntax Diagram

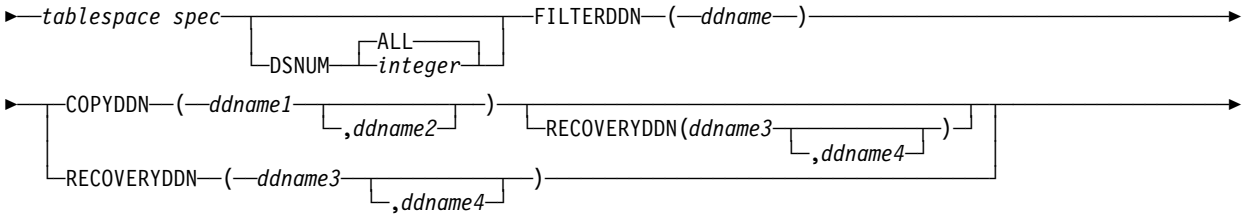
For guidance in interpreting syntax diagrams, see “How to Read the Syntax Diagrams” on page 1-3.



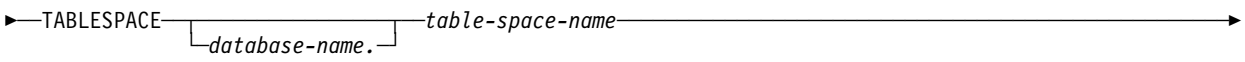




DFSMS Concurrent Copy with FILTER spec:



tablespace 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 2-7.

TABLESPACE
Specifies the table space (and, optionally, the data base 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.

COPY cannot copy a table 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, the job terminates and you receive error message DSNU419I.

DSNUM
Identifies a partition or data set, within the table space, that is to be copied; or it copies the entire table space.

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

ALL
Copies the entire table space.

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

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

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

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.

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 *ddnamex*

Specifies the DD statements for the primary and backup copied data sets for the image copy at the local site.

ddnamex is the DD name.

The **default** is **SYSCOPY** for the primary copy.

| If you use the CHANGELIMIT REPORTONLY option, you may use a DD
| DUMMY card when you specify the SYSCOPY output data set. This card pre-
| vents 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=(,CATLG). 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 table spaces in the list.

RECOVERYDDN *ddnamex*

Specifies the DD statements for the primary and backup copied data sets for the image copy at the recovery site.

ddnamex 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 table spaces in the list.

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 from a single data set, if you used DSNUM).

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

NO
Makes only an incremental image copy. Only changes since the last image copy are copied.

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 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 Concurrent option. COPY uses this data set to automatically build a list of table spaces to be copied by DFSMS/DSS with one DFSMS/DSS "DUMP" statement.

If FILTERDDN is specified, the SYSIBM.SYSCOPY records for all table spaces 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 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.

When SHRLEVEL CHANGE is specified, 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 32 KB page size.

CONCURRENT

Invokes 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.

When you use CONCURRENT option, 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

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 value from 0 to 100.

percent_value2

Specifies the second value in the CHANGELIMIT range. *percent_value2* must be an integer value from 0 to 100.

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)*.

REPORTONLY
Specifies that image copy information is displayed. If the REPORTONLY option is specified, then only image copy information is displayed. Image copies are not taken, only recommended.

Sample JCL and Control Statements

In some cases, a COPY utility job could 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 2-72 for further information.

Examples

Example 1: Full Image Copy: Make a full image copy of table space DSN8S51E in database DSN8D51A.

```
//SYSCOPY  DD DSN=COPY001F.IFDY01,UNIT=SYSDA,VOL=SER=CPY01I,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//SYSIN DD *
    COPY TABLESPACE DSN8D51A.DSN8S51E
```

Example 2: Copies for Local Site and Recovery Site: Make full image copies of table space DSN8S51C in database DSN8D51P at the local site and the recovery site.

```
//COPY1    DD DSN=COPY001D.IFDY01,UNIT=SYSDA,VOL=SER=CPY001,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY2    DD DSN=COPY002D.IFDY01,UNIT=SYSDA,VOL=SER=CPY002,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY3    DD DSN=COPY003T.IFDY01,UNIT=TAPE,VOL=SER=CPY03T,
//         DISP=(NEW,CATLG,CATLG),LABEL=(1,SL)
//COPY4    DD DSN=COPY004T.IFDY01,UNIT=TAPE,VOL=SER=CPY04T,
//         DISP=(NEW,KEEP,KEEP),LABEL=(1,SL)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
    COPY TABLESPACE DSN8D51P.DSN8S51C
    COPYDDN     (COPY1,COPY2)
    RECOVERYDDN (COPY3,COPY4)
```

Example 3: Incremental Copy with Updates Allowed: Make incremental image copies of table space DSN8S51D in database DSN8D51A, allowing update activity to occur during the copy process.

```
//SYSCOPY  DD DSN=COPY001I.IFDY01,UNIT=SYSDA,VOL=SER=CPY01I,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY2    DD DSN=COPY002I.IFDY01,UNIT=SYSDA,VOL=SER=CPY02I,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY3    DD DSN=COPY003I.IFDY01,UNIT=SYSDA,VOL=SER=CPY03I,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY4    DD DSN=COPY004I.IFDY01,UNIT=SYSDA,VOL=SER=CPY04I,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//SYSIN DD *
    COPY TABLESPACE DSN8D51A.DSN8S51D
    COPYDDN      (SYSCOPY,COPY2)
    RECOVERYDDN  (COPY3,COPY4)
    FULL NO
    SHRLEVEL CHANGE
```

Example 4: Invoking DFSMS Concurrent Copy with the COPY Utility: Copy a table space, using the CONCURRENT option to invoke DFSMS Concurrent Copy. Use a DSSPRINT DD card for message output.

```
//COPY     EXEC DSNUPROC,SYSTEM=V51A
//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
//SYSUT1   DD DSN=SYSUT1.L141S2,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
    COPY TABLESPACE DBASE1AA.TABLESPC
    COPYDDN     (SYSCOPY1)
    CONCURRENT
```

Example 5: 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)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
    COPY TABLESPACE TS1
         TABLESPACE TS2
         TABLESPACE TS3
    FILTERDDN(FILT)
    COPYDDN(SYSCOPY)
    CONCURRENT
    SHRLEVEL REFERENCE
```

Example 6: Invoking DFSMS Concurrent Copy with a List: Copy a list of table spaces, using the CONCURRENT option to invoke DFSMS Concurrent Copy. Allow update activity during the COPY operation.

```
//UTILSAMP JOB USER=SYSADM,PASSWORD=SYSADM,NOTIFY=SYSADM,
//         REGION=4M,TIME=1440,MSGCLASS=A,CLASS=A
//*
//UTILSAMP EXEC PGM=DSNUTILB,REGION=1024K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*
//*********************************************************************
//*  DATA SETS USED BY THE UTILITY                                    *
//*********************************************************************
//SYSCOPY1  DD DSN=CONCOPY1.MEMBER4.STEP1.SYSCOPY1,
//             DISP=(MOD,CATLG,DELETE),UNIT=SYSDA,
//             SPACE=(CYL,(42,5),RLSE)
//SYSCOPY2  DD DSN=CONCOPY1.MEMBER4.STEP1.SYSCOPY2,
//             DISP=(MOD,CATLG,DELETE),UNIT=SYSDA,
//             SPACE=(CYL,(42,5),RLSE)
//SYSCOPY3  DD DSN=CONCOPY1.MEMBER4.STEP1.SYSCOPY3,
//             DISP=(MOD,CATLG,DELETE),UNIT=SYSDA,
//             SPACE=(CYL,(42,5),RLSE)
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD *
    COPY TABLESPACE DBAU2901.TPAU2901
         COPYDDN(SYSCOPY1)
         TABLESPACE DBAU2901.TLAU2902
         COPYDDN(SYSCOPY2)
         TABLESPACE DBAU2901.TSAU2903
         COPYDDN(SYSCOPY3)
         CONCURRENT SHRLEVEL CHANGE
```

Example 7: 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 DSN8D51P.DSN8S51C CHANGELIMIT(10,40) REPORTONLY
```

Example 8: 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 DSN8D51P.DSN8S51C CHANGELIMIT(5)
```

Chapter 2-7. 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.

Syntax Diagram: For diagram of DIAGNOSE syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 2-81.

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.

Instructions for Running DIAGNOSE

In order to run DIAGNOSE, you must:

1. Prepare the necessary data sets, as described in “Creating JCL Statements for Required Data Sets” on page 2-80.
2. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for DIAGNOSE, see “Sample JCL and Control Statements” on page 2-84.)
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 2-80. (For a complete description of the syntax and options for DIAGNOSE, see “Syntax and Options of the Control Statement” on page 2-81.)

Invoking DIAGNOSE

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for an explanation of ways to invoke DB2 utilities.

Creating JCL Statements for Required Data Sets

Table 13 describes the data sets used by DIAGNOSE. Include statements in your JCL for each required data set.

Table 13. 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

For the task described, specify the options and values shown with your utility control statement. The following task is described here:

"Forcing a Utility Abend with DIAGNOSE"

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

You can terminate DIAGNOSE with the TERM UTILITY command.

Concurrency and Compatibility

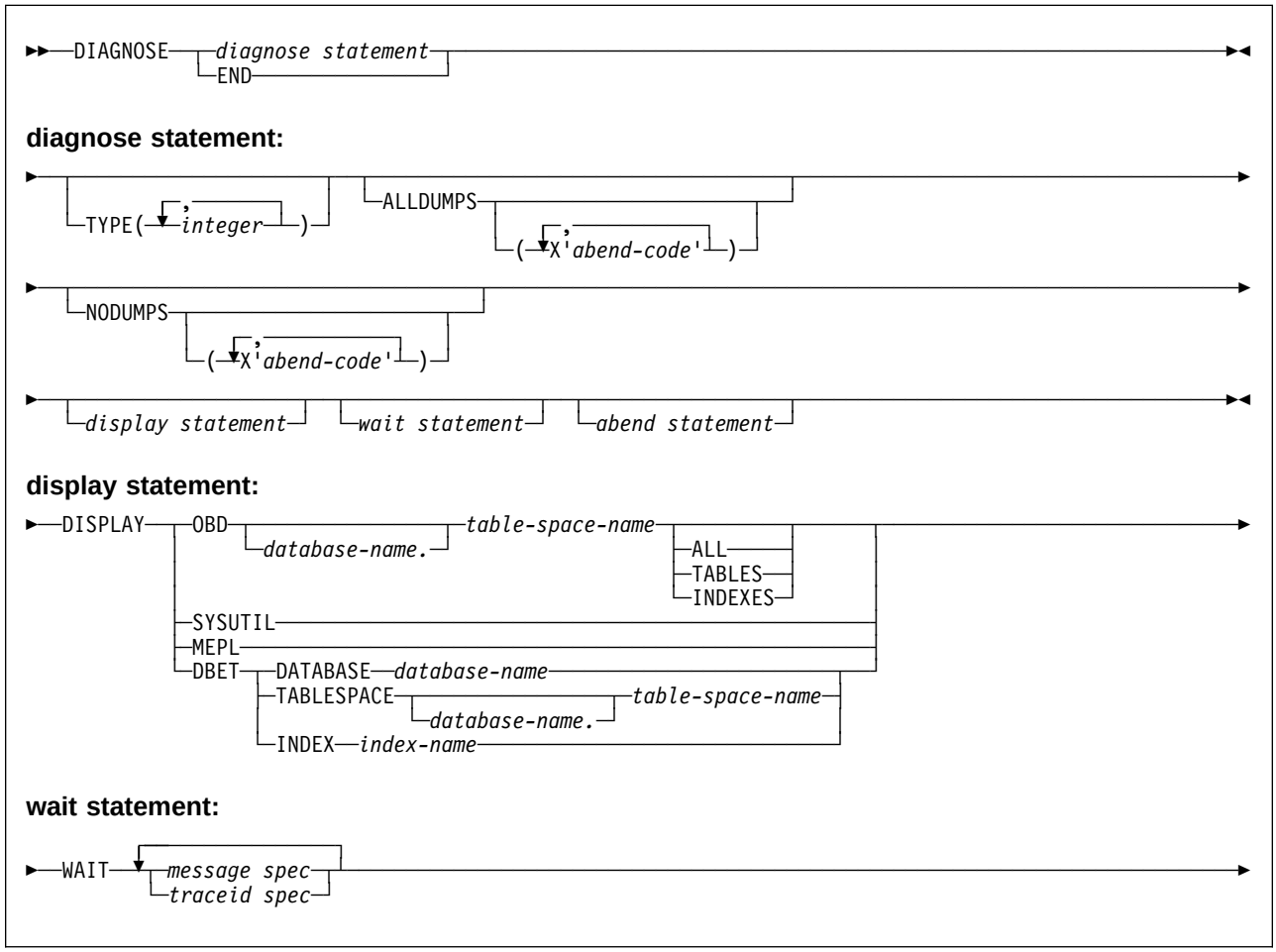
DIAGNOSE can run concurrently on the same target object with any SQL operation or utility, except a utility running on DSNDB01.SYSUTILX.

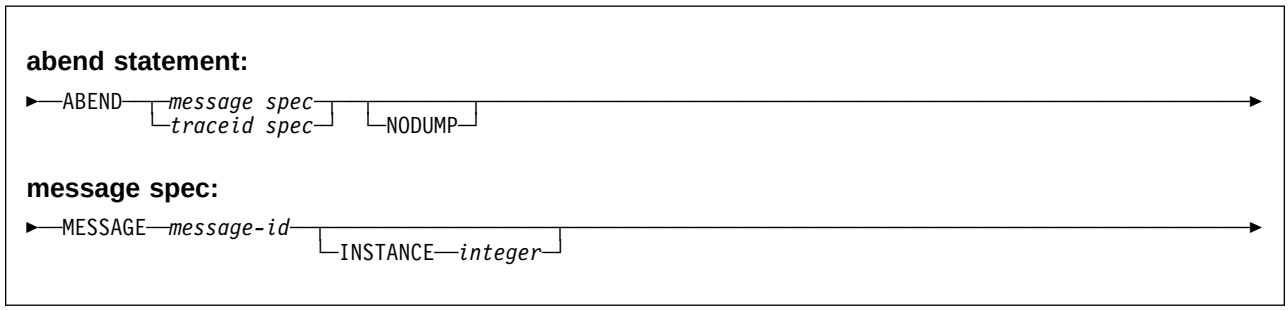
Syntax and Options of the Control Statement

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.

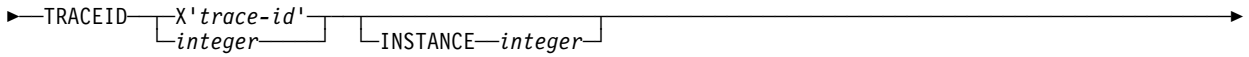
Syntax Diagram

For guidance in interpreting syntax diagrams, see “How to Read the Syntax Diagrams” on page 1-3.





traceid 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 2-7.

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 *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.

WAIT
Upon 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
Upon 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.

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.

MESSAGE *message-id*
Specifies a DSNU*xxx* 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 *Messages and Codes*.

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

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.

Sample JCL and Control Statements

Examples

Example 1: Sample JCL for DIAGNOSE:

```
//UTILSAMP JOB USER=SYSADM,PASSWORD=SYSADM,NOTIFY=SYSADM,
//         REGION=4M,TIME=1440,MSGCLASS=A,CLASS=A
/*ROUTE PRINT STLVM14.CARBAJAL
//*
//*        UTILITY SAMPLE JOB
//*
//UTILSAMP EXEC PGM=DSNUTILB,REGION=4096K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*
//*********************************************************************
//*  DATA SETS USED BY THE UTILITY                                    *
//*********************************************************************
//SYSCOPY1 DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *

 DIAGNOSE  ABEND MESSAGE U400
                 INSTANCE 1
                 NODUMP
                    COPY TABLESPACE DSN8D51A.DSN8S51E
                      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 2-8. 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.

Syntax Diagram: For a diagram of LOAD syntax and a description of available options, see "Syntax and Options of the Control Statement" on page 2-113.

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 *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.

The LOAD utility cannot be run on the DSNDB01 or DSNDB06 databases, except it can be used to add lines to SYSIBM.SYSSTRINGS and SYSIBM.SYSPROCEDURES.

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

Instructions for Running LOAD

In order to run LOAD, you must:

1. Read "Before Running LOAD" on page 2-92 in this chapter.
2. Prepare the necessary data sets, as described in "Creating JCL Statements for Required Data Sets" on page 2-89.
3. Create JCL statements, by using one of the methods described in "Chapter 2-1. Invoking DB2 Online Utilities" on page 2-7. (For examples of JCL for LOAD, see "Sample JCL and Control Statements" on page 2-132.)
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 2-92. (For a complete description of the syntax and options for LOAD, see "Syntax and Options of the Control Statement" on page 2-113.)

5. Check the compatibility table in “Concurrency and Compatibility” on page 2-112 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” on page 2-110.
7. Read “After Running LOAD” on page 2-103 in this chapter.

Invoking LOAD

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for an explanation of ways to invoke DB2 utilities.

Phases and Data Flow

Table 14 describes the phases of the LOAD utility.

Table 14 (Page 1 of 2). LOAD Phases

Phase	Description
UTILINIT	Initialization and setup.
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 SORTKEYS is used, 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.
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.

Table 14 (Page 2 of 2). LOAD Phases

Phase	Description
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.

Creating JCL Statements for Required Data Sets

Table 15 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 15 (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]
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]
SORTLIB	Contains the DFSORT load module. If you omit this, and sorting is needed, SYS1.SORTLIB is dynamically allocated.	No
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.	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

Table 15 (Page 2 of 2). Data Sets Used by LOAD

Data Set	Description	Required?
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 objects are named in the utility control statement and do not require DD cards 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 16 to calculate the size of work data sets for LOAD.

Table 16. 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 + 13, longest foreign key + 13)* × *(number of keys*
| *extracted)*

Calculating the number of keys extracted:

- Count 1 for each index.
- Count 1 for each foreign key that is not exactly indexed (that is, where foreign key and index definitions do not correspond identically).
- For each foreign key that is exactly indexed (that is, where foreign key and index definitions correspond identically):
  - Count 0 for the first relationship in which the foreign key participates.
  - Count 1 for subsequent relationships in which the foreign key participates (if any).
- Multiply count by the number of rows to be loaded.

| **Calculating the foreign key: f**

| *max(longest foreign key + 13)* × *(number of keys extracted)*

Calculating the map: m

| The data set must be large enough to accommodate 1 map entry (length = 17
| 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 = 84 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 needed.

Allocating twice the space used by the input data sets is usually adequate for the sort work data sets. One or two large SORTWK*nn* data sets are preferable to

several small ones. For further information, see *DFSORT Application Programming: Guide.*

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.

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 could 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 upon 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 could 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).

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 12 is an example that replaces one table in a single-table table space:

```
LOAD DATA
  REPLACE
  INTO TABLE DSN8510.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 12. 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 DSN8510.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 13. 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 DSN8510.TDSPTXT
  ( DSPINDEX  POSITION (2)     CHAR(2),
    LINENO    POSITION (6)     CHAR(2),
    DSPLINE   POSITION (80)    CHAR(79) )
```

Figure 14. 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 DSN8510.TDSPTXT without changing any data in DSN8510.TOPTVAL. To do this:

1. Delete all the rows from DSN8510.TDSPTXT using an SQL DELETE statement. (The mass delete works most quickly on a segmented table space.)

   ```
   DELETE FROM DSN8510.TDSPTXT;
   ```

2. Use the LOAD job in Figure 15 to replace the rows in that table.

```
LOAD DATA CONTINUEIF(72:72)='X'
  RESUME YES
  INTO DSN8510.TDSPTXT
  ( DSPINDEX  POSITION (2)     CHAR(2),
    LINENO    POSITION (6)     CHAR(2),
    DSPLINE   POSITION (80)    CHAR(79) )
```

Figure 15. 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.

The REPLACE and RESUME options can be specified 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 DSN8510.EMP PART 1 REPLACE WHEN (1) = '0'
  ( EMPNO     POSITION (1:6)  CHAR(6),
    FIRSTNME  POSITION (7:18) CHAR(12),

  ⋮
                                       )
  INTO TABLE DSN8510.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 DSN8510.EMP REPLACE PART 1

LOAD DATA INDDN EMPLDS2 CONTINUEIF(72:72)='X'
  RESUME YES
  INTO TABLE DSN8510.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 DSN8510.EMP PART 2 REPLACE
```

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 *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; later, 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 tells 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 that status, and typically you run the CHECK utility to remove them; see “Resetting the Check Pending Status” on page 2-104.

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 *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. CHECK DATA checks a complete referential structure, while 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 needed to build a dictionary is dependent on the frequency of patterns in the data. For large data sets, the number of rows needed 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 2-14. REORG” on page 2-197 and “Chapter 2-17. RUNSTATS” on page 2-277 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 16.

```
LOAD DATA
  REPLACE KEEPDICTIONARY
  INTO TABLE DSN8510.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 16. 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 (DPROP) licensed program. DPROP 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 DPROP, you do not need to extract all the data in a database or data set. You use a statement such as an SQL subselect to 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 DPROP 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. DPROP 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 DPROP 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 DPROP) 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 DPROP include the CONTINUEIF option to describe the extracted data to DB2 LOAD.

In the second case, you can have DPROP name the data set containing the extracted data in the SYSREC DD statement in the LOAD job. (In that case, DPROP makes no provision for transmitting the extracted data across a network.)

Normally, you do not have to edit the job statements produced by DPROP. 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.)

DPROP 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

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 REF, 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.
- Load numeric data in its internal representation.
- Avoid data conversion, such as integer to decimal or decimal to floating-point.
- Specify LOG NO on the LOAD statement, and then take a full image copy with SHRLEVEL REFERENCE .
- When you specify LOAD REPLACE, specify LOG NO with COPYDDN 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.

DASD space requirements for the SYSUT1 and SORTOUT data sets are also reduced, especially if an estimate of the number of keys to sort is provided.

Also, elapsed time from the start of the reload phase to the end of the build phase is reduced.

However, if the index keys are already in sorted order, or there are no indexes, SORTKEYS does not provide any advantage.

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:

- Count 1 for each index
- Count 1 for each foreign key where foreign key and index definitions are not identical
- For each foreign key where foreign key and index definitions are identical:
  - Count 0 for the first relationship in which the foreign key participates
  - Count 1 for subsequent relationships in which the foreign key participates (if any).
- 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 SORTKEYS is used and an estimate of the number of keys to be sorted is omitted or specified as 0, the sort input data set (SYSUT1) and the sort output data set (SORTOUT) are required. See page 2-90 for instructions on calculating the size of those data sets.

Improving Performance with 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, PREFORMAT might be a technique to consider. This technique will only be of value if DB2's preformatting is causing a measurable delay with the INSERT processing or causing inconsistent elapsed times for INSERT applications. It is recommended that a performance assessment be conducted before and after 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 tablespace during execution time. This may eliminate execution time delays but adds the preformatting cost as setup prior to the application's execution. PREFORMAT primes a new tablespace and prepares it for INSERT processing. Once the preformatted space is utilized and DB2 has to extend the tablespace, normal dataset 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 preallocated prior to the application's execution. This would be the case for a table that acts as a holding tank for work items coming into a system that are later used to feed a backend task that processes the work items.

Preformatting of a tablespace containing a table used for query processing may cause table space scans to read additional empty pages, extending the elapsed time for these queries. 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 datasets or have DB2 manage the datasets. For user-managed datasets, DB2 will not delete and reallocate them during utility processing. The size of the dataset will not shrink back to the original dataset allocation size but will either remain the same or increase in size if additional space or data is added. This has implications when PREFORMAT is used since preformatting will cause 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 datasets, DB2 will delete and reallocate them if REPLACE is specified on the LOAD, or if REORG is run. This will result in the datasets 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 PREFORMAT option with DB2 managed datasets will at minimum cause the full primary allocation amount of a dataset to be preformatted following the reload of data into the tablespace.

For both user-managed and DB2 managed datasets, if the dataset 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: PREFORMAT can eliminate dynamic preformatting delays when inserting into a new tablespace. 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 since empty preformatted pages will be read. It is best to use the PREFORMAT option for tablespaces that start out empty and are filled through high insert activity before any query access is performed against the tablespace. Mixing inserts and non-indexed queries against a preformatted tablespace 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.

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 image copies taken after the LOAD completes are not necessary.

Use the RUNSTATS utility so that the DB2 catalog statistics take into account the newly loaded data, and SQL paths can be selected 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:

- REPAIR SET with NOCOPYPEND
- LOAD REPLACE LOG YES

- LOAD REPLACE LOG NO with an inline copy
- REORG LOG YES
- REORG LOG NO with an inline copy

If you use LOG YES and do not make an image copy of the table space, later 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 Recovery Pending Status

LOAD places all the index spaces for a table space in the “recovery pending” status if you end the job (using -TERM UTILITY) before it completes the INDEXVAL phase. It places the table space itself in “recovery pending” if you end the job before it completes the RELOAD phase.

There are two variations of recovery pending set by LOAD:

- Recovery pending (RECP in DISPLAY output) means an index, table space, or partition of an index or table space is in recovery pending status.
- Page set recovery pending (PSRCP in DISPLAY output) means that an entire type 1 index is in recovery pending status.

Resetting the recovery pending status depends on when the utility terminated:

- If the data is intact (-DISPLAY shows indexes are in recovery pending status but not the table space), you can recover the indexes using RECOVER INDEX. If the index is in page set recovery pending (PSRCP), you must recover the entire index. However, for partitioned indexes and for nonpartitioned indexes in recovery pending (RECP), you can use the PART clause of RECOVER INDEX to recover separate partitions of the index.
- If the data is not intact (-DISPLAY shows the table space to be in recovery 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. If you recover it, RECOVER places all indexes in recovery 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 no invalid 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 2-39.)

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 statement runs CHECK DATA against the table space containing the project activity table. It assumes the existence of exception tables named DSN8510.EPROJACT and DSN8510.EEPA.

```
CHECK DATA TABLESPACE DSN8D51A.PROJACT
   DELETE YES
   FOR EXCEPTION IN DSN8510.PROJACT USE DSN8510.EPROJACT
                 IN DSN8510.EMPPROJACT USE DSN8510.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 could not delete any parent rows from the project table, and so could 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 statement runs CHECK DATA against the table space containing the project activity table after LOAD RESUME:

```
CHECK DATA TABLESPACE DSN8D51A.PROJACT
   SCOPE PENDING
   DELETE YES
   FOR EXCEPTION IN DSN8510.PROJACT USE DSN8510.EPROJACT
                 IN DSN8510.EMPPROJACT USE DSN8510.EEPA
   SORTDEVT SYSDA
   SORTNUM 4
```

As before, the JCL for the job needs DD statements to define data sets for the error and sort data sets.

Running RUNSTATS after Loading Any Table

The RUNSTATS utility gathers data about space use and row clustering to update the DB2 catalog. The data is used to select access paths when executing SQL statements. To ensure that information is available and current, run RUNSTATS after loading a table space and before binding any application packages or plans that access it.

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 recover an index that is inconsistent with its data, use the RECOVER 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.

Considerations for Running LOAD

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

Converting Input Data

The LOAD utility converts data between compatible data types. Tables 17, 18, 19, and 20 identify the allowable data conversions and the defaults used when the input data type is not specified in a field specification of the INTO TABLE statement.

Table 17. 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 18 (Page 1 of 2). Character Data Conversion

Input Data Types	Output Data Types		
	CHAR	**VARCHAR**	**LONG VARCHAR**
CHAR	Default	Allowed	Allowed

Table 18 (Page 2 of 2). Character Data Conversion

Input Data Types	Output Data Types		
	CHAR	**VARCHAR**	**LONG VARCHAR**
VARCHAR	Allowed	Default	Default

Table 19. Time Data Conversion

Input Data Types	Output Data Types		
	DATE	**TIME**	**TIMESTAMP**
DATE EXTERNAL	Default		
TIME EXTERNAL		Default	
TIMESTAMP EXTERNAL	Allowed	Allowed	Default

Table 20. Graphic Data Conversion

Input Data Types	Output Data Types		
	GRAPHIC	**VARGRAPHIC**	**LONG VARGRAPHIC**
GRAPHIC	Default	Allowed	Allowed
VARGRAPHIC	Allowed	Default	Default

Input fields with data types CHAR, CHAR MIXED, 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.

Conversion errors cause LOAD:

- To abend, if there is no DISCARDS processing
- To map the input record for later 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 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.

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 Recovery Pending Status

You cannot load records specifying RESUME YES if any partition is in the recovery pending status. In addition, you cannot load records if any index on the table being loaded is in the recovery pending status. See “Chapter 2-13. RECOVER TABLESPACE” on page 2-173 for information about resetting the recovery pending status.

If you are replacing a partition, these restrictions are relaxed; the partition being replaced and its corresponding index partition can be in the recovery pending status. However, all nonpartitioned indexes must *not* be in the page set recovery pending status.

There are three recovery pending restrictive statuses:

RECP The table space, index space, or partition is in a recovery 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 can be reset by recovering only the single logical partition.

RECP* The logical partition of the type 2 nonpartitioned index is in a recovery pending status. The entire nonpartitioned index is inaccessible. RECP* can be reset by recovering only the single logical partition.

PSRCP The type 1 index is in a recovery pending status. To recover, the entire page set must be recovered. Recovery of a logical partition is prohibited.

Using Exit Procedures

Any field procedure associated with a column of a table being loaded is invoked to encode the data before it is loaded. The field procedures for all columns are invoked before any edit or validation procedure for the row.

Any field specification that describes the data is checked before a field procedure is invoked. That is, the field specification must describe the data as it appears in the input record.

Terminating or Restarting

For instructions on restarting a utility job, see “Restarting an Online Utility” on page 2-28.

Terminating LOAD: If LOAD is terminated by the TERM UTILITY command during the reload phase, the records are not erased. The table space and indexes remain in recovery pending status.

If LOAD is terminated by the TERM UTILITY command during the sort or build phases, then the indexes not yet built remain in the recovery pending status.

If SORTKEYS is used and the LOAD job terminates during the RELOAD, SORT, or BUILD phase, then both RESTART and RESTART(PHASE) restart from the beginning of the RELOAD phase.

Table 21. LOAD Phases and Pending Statuses

Phase	Effect on Pending Status
Reload	Places table space in recovery pending status, then resets the status. Places indexes in recovery 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 recovery 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 recovery pending status for unique indexes.
Enforce	Resets check pending status for table space.

Restarting LOAD: Table 22 on page 2-111 provides information about restarting LOAD, depending on the phase LOAD was in when the job stopped.

- If LOAD is restarted in the UTILINIT phase, it is re-executed from the beginning of the phase.
- If LOAD abends or system failure occurs while it is in the UTILTERM phase, it must be restarted with RESTART(PHASE).

In this table, the TYPE column distinguishes between the effects of specifying RESTART(CURRENT) or RESTART(PHASE).

Table 22. 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
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 2-8. LOAD” on page 2-87 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 upon the data sets used and whether any input data sets have been rewritten.
5. LOAD RESUME YES cannot be restarted in the BUILD 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 backout. 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.

You can restart LOAD at its last commit point or at the beginning of the phase during which operation ceased. The phases that have completed are identified in

LOAD output messages; the specific phase during which operation stopped can be identified with the DISPLAY command.

Restarting After an Out Of Space Condition

See “Restarting After an Out of Space Condition” on page 2-29 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. However, if a type 1 nonpartitioned index exists on a partitioned table space, utilities operating on different partitions of a table space can be incompatible because of contention on the nonpartitioned index.

For nonpartitioned type 2 indexes, LOAD PART:

- Drains only the logical partition
- Does not set the page set recovery pending status (PSRCP)
- Does not respect PCTFREE or FREEPAGE attributes when inserting keys

Claims and Drains

Table 23 shows which claim classes LOAD drains and the restrictive states the utility sets.

Table 23. LOAD. Use of claims and drains; restrictive states set.

Target	LOAD	LOAD PART
Table space, index, or physical partition of a table space or index	DA/UTUT	DA/UTUT
Nonpartitioned type 2 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
- MODIFY STATISTICS
- REPORT
- STOSPACE.

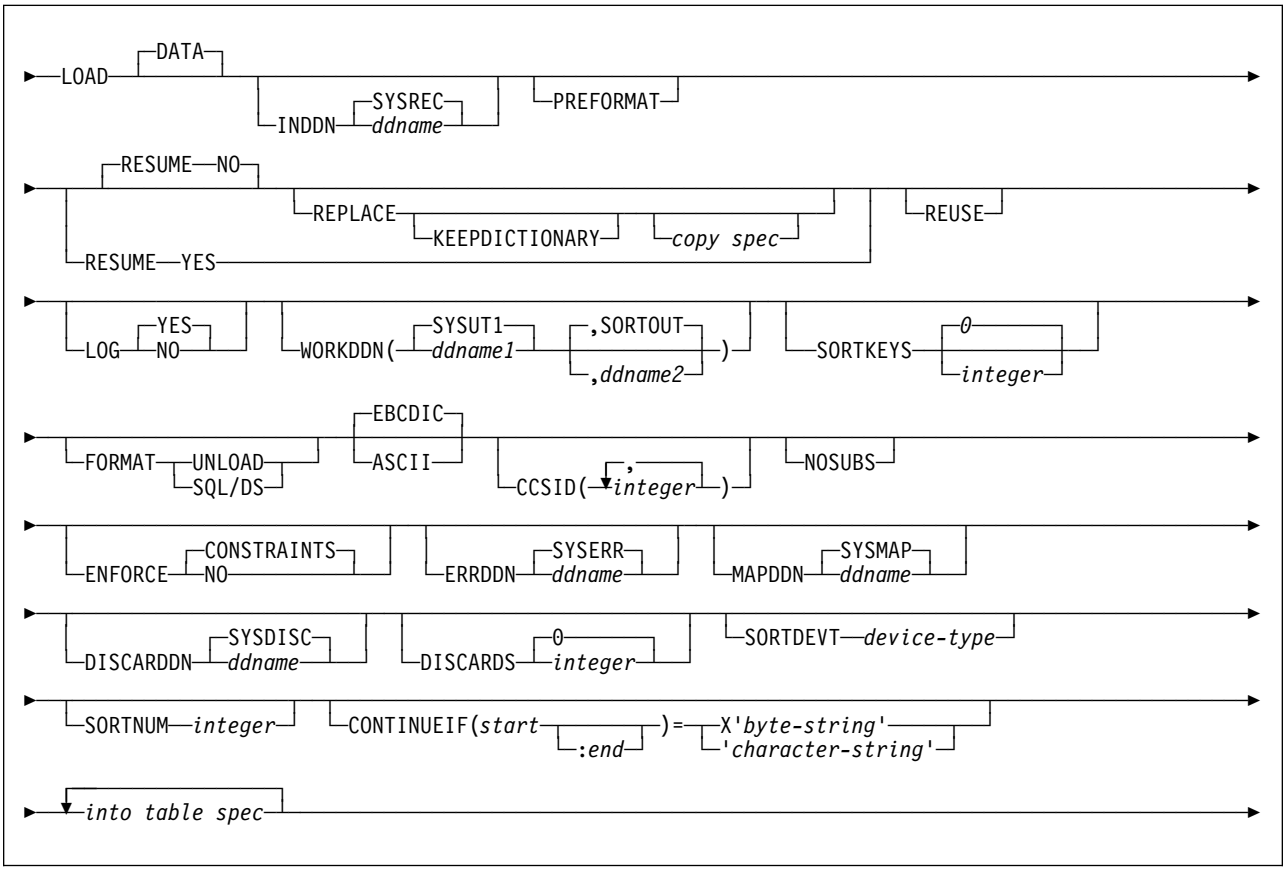
SQL operations and other online utilities on the same target partition are incompatible.

Syntax and Options of the Control Statement

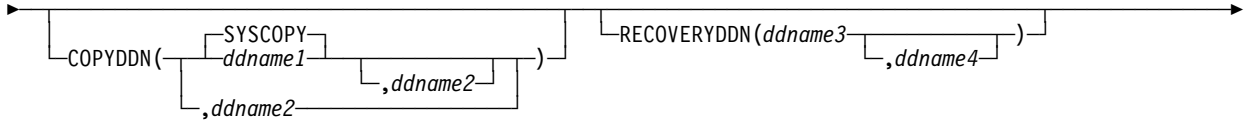
The utility control statement defines the function the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name 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 1-3.



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 2-7.

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 2-121 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 indexspaces 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 2-123 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, 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. 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 *Administration Guide*.

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 2-121 for information about specifying REUSE for individual partitions.

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 2-103.

If the table space has a TYPE 2 nonpartitioning index, some logging might occur in the build phase if DB2 is accessing another partition for an SQL statement or utility. Also, DB2 logs index structure changes to allow recoverability of the nonpartitioning index in case an abend occurs.

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 with the reload and build phases 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. See “Improving Performance with SORTKEYS” on page 2-101.

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 2-123.

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 *SQL/Data System Data Base Services Utility for VM/System Product.*

LOAD cannot load SQL/DS strings that are longer than the DB2 limit. For information about DB2 limits, see “Limits in DB2 for OS/390” on page X-3.

SQL/DS data that has been unloaded to disk under VM/370 resides in a simulated OS/VS 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 VM/370 FILEDEF must define it as an OS/VS 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.

| **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 terminates with return code 8.

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.

byte-string
Is a string of hexadecimal digits. 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 could 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 could be

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

| **COPYDDN** *ddnamex*
| Specifies the DD statements for the primary and backup copied data sets for the image copy.

| *ddnamex* 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, since these image copies might contain unique index vio-
lations or referential constraint violations.

RECOVERYDDN *ddnamex*
Specifies the DD statements for the primary and backup copied data sets for the image copy at the recovery site.

ddnamex is the DD name.

You cannot have duplicate image copy data sets. The same rules apply for RECOVERYDDN and COPYDDN.

INTO TABLE Spec

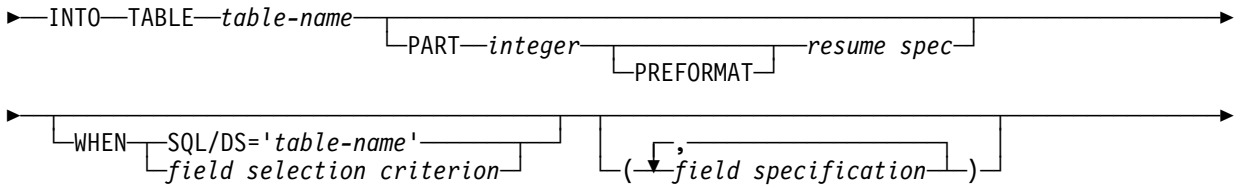
More than one table or partition for each table space can be loaded with a single invocation of the LOAD utility. At least one INTO TABLE statement is required for each table to be loaded, to:

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

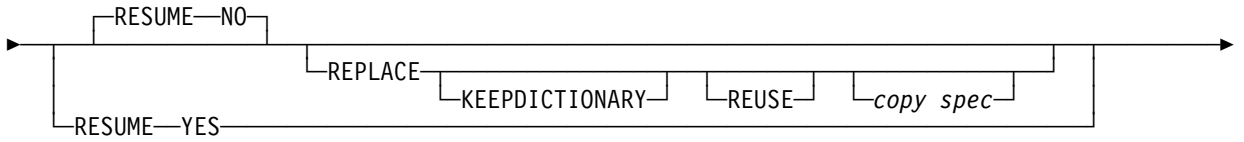
All tables specified by INTO TABLE statements must belong to the same table space.

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.

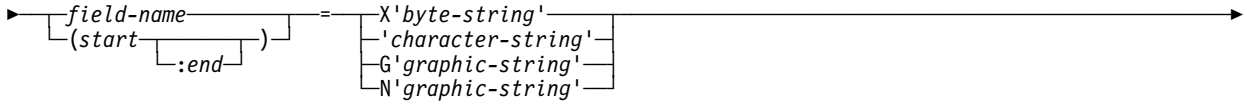
into table spec:



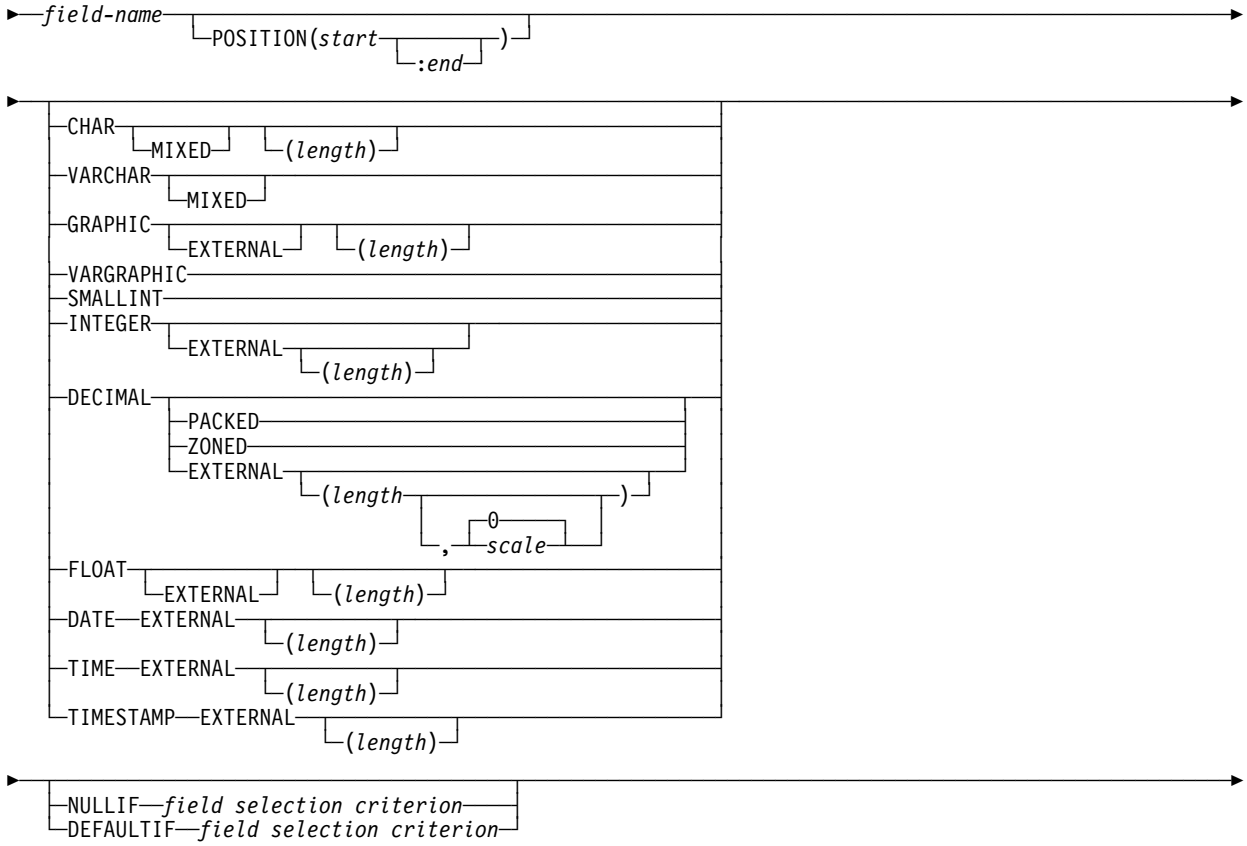
resume spec:



field selection criterion:



field specification:



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 2-7.

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.

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 digits. 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. G can be specified 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. N can be specified 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.
- 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.

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 2-135.

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 or VARGRAPHIC, 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.

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 24 shows the default length, in bytes, for each data type.

Table 24. Default Length of Each Data Type (in bytes)

Data Type	Length in Bytes
CHARACTER	Length used in column definition
DATE	10 (or installation default)
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
SMALLINT	2
TIME	8 (or installation default)
TIMESTAMP	26

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.

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.

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 *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*.

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.

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

You can also specify DEC EXTERNAL and DEC EXTERNAL(*length*).

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.

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 *SQL Reference*.

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 following 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 not specified, or is between 22 and 53 inclusive, the number is 64 bits in the following 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.

You can also specify REAL for single precision floating point and DOUBLE PRECISION for double precision floating point.

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.

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 *SQL Reference*.

You can also specify INT EXTERNAL.

INTEGER

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

You can also specify INT.

SMALLINT

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

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 *SQL Reference* for more information about the timestamp data type.

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.

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.

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 2-125. 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’.

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 2-125. 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'.

Sample JCL and Control Statements

Examples

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.

```
//UTILSAMP EXEC PGM=DSNUTILB,REGION=1024K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*
//*****************************************************************
//*  DATA SETS USED BY THE UTILITY                                *
//*****************************************************************
//SORTOUT  DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTWK01 DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSDISC  DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSERR   DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSMAP   DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSRECAC DD  DSN=DSN510.SDSNSAMP(DSN8LAC),
//             DISP=SHR
//SYSUT1   DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSPRINT DD  SYSOUT=*
//UTPRINT  DD  SYSOUT=*
//SYSIN    DD  *

 LOAD DATA INDDN(SYSRECAC) RESUME YES
      INTO TABLE DSN8510.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 17 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 DSN8510.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 17. 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 18 on page 2-134 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 18. Records in an Input Data Set for LOAD

Table 25 shows how the same records appear if you then execute the statement SELECT * FROM DSN8510.DEPT under SPUFI.

Table 25. 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 DSN8510.EMP
```

Example 4: Load Data into Two Tables: Load data from the data set specified by the EMPLDS DD statement into the DSN8510.EMP and SMITH.EMPEMPL tables.

```
LOAD DATA INDDN EMPLDS
  INTO TABLE DSN8510.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 DSN8510.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 DSN8510.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 DSN8510.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 DSN8510.PROJ table. The table space containing the DSN8510.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 DSN8510.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 DSN8510.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 DSN8510.TOPTVAL table. The table space that contains DSN8510.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 DSN8510.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 DSN8510.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 DSN8510.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 DSN8510.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 DSN8510.DEPT table. This table has 3 indexes and two foreign keys. The foreign key and index definitions are identical, so the calculation described in “Improving Performance with SORTKEYS” on page 2-101 is:

```
(3 + 0) * 22,000 = 66,000
```

LOAD statement:

```
LOAD DATA REPLACE INDDN INPUT
 SORTKEYS 66000
 CONTINUEIF(79:80)='++'
 INTO TABLE DSN8510.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 DSN8510.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.

Chapter 2-9. 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.

Syntax Diagram: For a diagram of MERGECOPY syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 2-144.

Output: Output from the MERGECOPY utility consists of one of the following:

- 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.

Instructions for Running MERGECOPY

In order to run MERGECOPY, you must:

1. Prepare the necessary data sets, as described in “Creating JCL Statements for Required Data Sets” on page 2-140.
2. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for MERGECOPY, see “Sample JCL and Control Statements” on page 2-147.)
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 2-141. (For a complete description of the syntax and options for MERGECOPY, see “Syntax and Options of the Control Statement” on page 2-144.)
4. Check the compatibility table in “Concurrency and Compatibility” on page 2-144 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” on page 2-143.

Invoking MERGECOPY

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for an explanation of ways to invoke DB2 utilities.

Phases and Data Flow

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.

Creating JCL Statements for Required Data Sets

Table 26 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 26. 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
Input data sets	These are image copy data sets which may be preallocated by the user. The DD names are defined by the user.	No

The following objects are named in the utility control statement and do not require DD cards 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. In order 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 this 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 2-144 for MERGECOPY syntax and option descriptions. See “Sample JCL and Control Statements” on page 2-147 for examples of MERGECOPY usage.

Instructions for Specific Tasks

For the tasks described, specify the options and values shown with your utility control statement. The following tasks are described here:

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 could 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” on page 2-99 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.

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

For instructions on restarting a utility job, see “Restarting an Online Utility” on page 2-28.

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 an Out of Space Condition” on page 2-29 for guidance in restarting MERGECOPY from the last commit point after receiving an out of space condition.

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.

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 type 1 nonpartitioned index exists on a partitioned table space, utilities operating on different partitions of a table space can be incompatible because of contention on the nonpartitioned index.

Table 27 shows the restrictive state the utility sets on the target object.

Table 27. MERGECOPY. Use of claims and drains; restrictive states set.

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
- MERGECOPY
- MODIFY RECOVERY
- RECOVER.

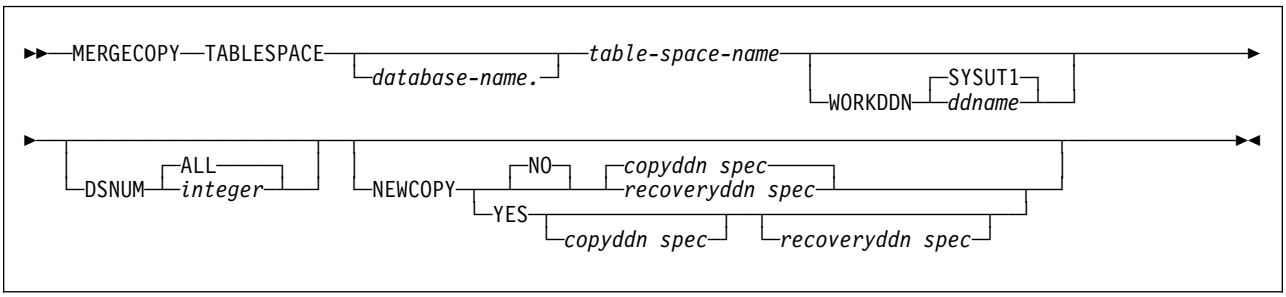
The target object can be a table space or partition.

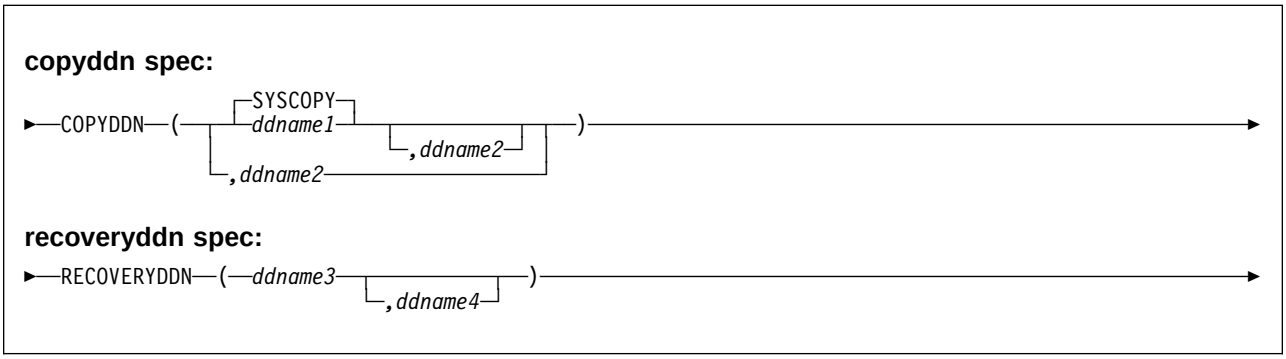
Syntax and Options of the Control Statement

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 1-3.





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 2-7.

TABLESPACE (*database-name.table-space-name***)**
Specifies the table space (and, optionally, the data base 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 inter-

mediate 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*n*

where: *n* 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.

Sample JCL and Control Statements

Examples

Example 1: Creating a Merged Incremental Copy: Create a merged incremental image copy of table space DSN8S51C.

```
//COPY1    DD DSN=COPY001I.IFDY01,UNIT=SYSDA,VOL=SER=CPY01I,
//         SPACE=(TRK,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY2    DD DSN=COPY002I.IFDY01,UNIT=SYSDA,VOL=SER=CPY02I,
//         SPACE=(TRK,(15,1)),DISP=(NEW,CATLG,CATLG)
//SYSIN DD *
    MERGECOPY TABLESPACE DSN8D51P.DSN8S51C
    COPYDDN      (COPY1,COPY2)
    NEWCOPY  NO
```

Example 2: Creating a Merged Full Image Copy: Create a merged full image copy of table space DSN8S51C.

```
//COPY1    DD DSN=COPY001F.IFDY01,UNIT=SYSDA,VOL=SER=CPY01I,
//         SPACE=(TRK,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY2    DD DSN=COPY002F.IFDY01,UNIT=SYSDA,VOL=SER=CPY02I,
//         SPACE=(TRK,(15,1)),DISP=(NEW,CATLG,CATLG)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
    MERGECOPY TABLESPACE DSN8D51P.DSN8S51C
    COPYDDN      (COPY1,COPY2)
    NEWCOPY  YES
```

Chapter 2-10. 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.

MODIFY should be run 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.

Syntax Diagram: For a diagram of MODIFY syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 2-153.

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 2-151.

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.

Instructions for running MODIFY

In order to run MODIFY you must:

1. Read “Before Running MODIFY” on page 2-151 in this chapter.
2. Prepare the necessary data sets, as described in “Creating JCL Statements for Required Data Sets” on page 2-150.

3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for MODIFY, see “Sample JCL and Control Statements” on page 2-154.)
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 2-151. (For a complete description of the syntax and options for MODIFY, see “Syntax Diagram” on page 2-153.)
5. Check the compatibility table in “Concurrency and Compatibility” on page 2-152 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 2-152.

Invoking MODIFY

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for an explanation of ways to invoke DB2 utilities.

Phases and Data Flow

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.

Creating JCL Statements for Required Data Sets

Table 28 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 28. 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 objects are named in the utility control statement and do not require DD cards 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 2-153 for MODIFY syntax and option descriptions. See “Sample JCL and Control Statements” on page 2-154 for examples of MODIFY usage.

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 Recovery Pending Status: You cannot run MODIFY RECOVER on a table space that is in recovery pending status. See “Chapter 2-13. RECOVER TABLESPACE” on page 2-173 for information about resetting the recovery pending status.

Instructions for Specific Tasks

For the tasks described, specify the options and values shown with your utility control statement. The following tasks are described here:

“Deleting SYSLGRNX Rows”
“Deleting All Image Copy Entries”
“Reclaiming Space in the DBD”

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 a return code of 4.

After the image copy information for a particular object is deleted from SYSIBM.SYSCOPY, the corresponding image copy cannot be used directly by the RECOVER TABLESPACE utility to restore the table space. If the full image copy data set is still available, DSN1COPY can be used to restore the table space to that image copy. Then RECOVER LOGONLY can be used to restore the table space to currency.

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

If you add a column to a table space and the record was previously fixed-length, DB2 treats the record as variable-length. As a result, the REORG utility decompresses the rows of the table space during the UNLOAD phase and then compresses them again during the RELOAD phase. To remedy this, use the following procedure:

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 returns the record to fixed-length status only if there are SYSCOPY rows to delete.

Terminating or Restarting MODIFY

For instructions on restarting a utility job, see “Restarting an Online Utility” on page 2-28.

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 2-28.

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 type 1 nonpartitioned index exists on a partitioned table space, utilities operating on different partitions of a table space can be incompatible because of contention on the nonpartitioned index.

Table 29 shows the restrictive state the utility sets on the target object.

Table 29. MODIFY. Use of claims and drains; restrictive states set.

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
- LOAD
- MERGECOPY
- MODIFY
- RECOVER
- REORG.

The target object can be a table space or partition.

MODIFY holds an X lock on DSNDB06.SYSCOPY for the duration of its allocation. As a result, any of the following utilities could time out when running concurrently with MODIFY:

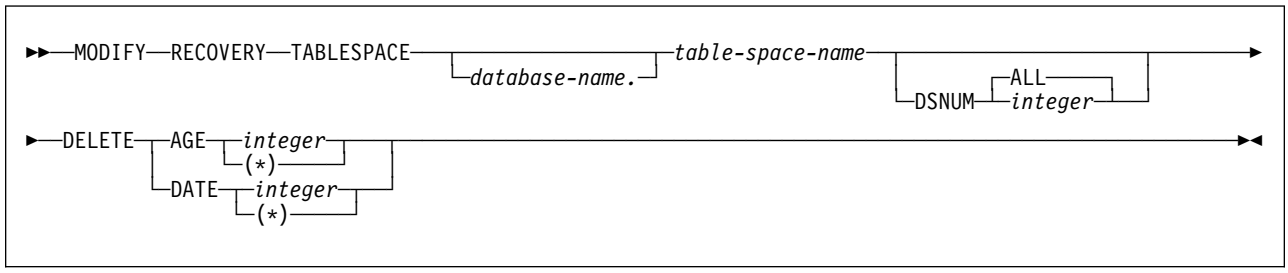
- COPY
- LOAD
- MERGECOPY
- MODIFY
- QUIESCE
- RECOVER
- REORG
- REPORT

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 1-3.



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 2-7.

TABLESPACE *database-name.table-space-name*
Specifies the database and the table space for which records are to be deleted.

database-name is the name of the database to which the table space belongs. *database-name* is optional.

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

table-space-name is 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*n*

where: *n* 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 could 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.

Sample JCL and Control Statements

Examples

Example 1: Delete SYSCOPY Records by Age: For the table space containing the employee table, delete all SYSCOPY records older than 90 days.

```
//UTILSAMP EXEC PGM=DSNUTILB,REGION=1024K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*
//*********************************************************************
//*  DATA SETS USED BY THE UTILITY                                    *
//*********************************************************************
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
    MODIFY RECOVERY TABLESPACE DSN8D51A.DSN8S51E DELETE AGE(90)
/*
```

Example 2: Delete SYSCOPY Records by Date: For the table space containing the department table, delete all SYSCOPY records written before 4 July 1985.

```
MODIFY RECOVERY TABLESPACE DSN8D51A.DSN8S51D
  DELETE DATE(850704)
```

Chapter 2-11. QUIESCE

The QUIESCE online utility establishes a quiesce point (the current log RBA) for a table space, partition, or list of table spaces, 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.

Syntax Diagram: For a diagram of QUIESCE syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 2-161.

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. (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.

Instructions for Running QUIESCE

In order to run QUIESCE, you must:

1. Read “Before Running QUIESCE” on page 2-158 in this chapter.
2. Prepare the necessary data sets, as described in “Creating JCL Statements for Required Data Sets” on page 2-158.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for QUIESCE, see “Sample JCL and Control Statements” on page 2-162.)
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 2-158. (For a complete description of the syntax and options for QUIESCE, see “Syntax and Options of the Control Statement” on page 2-161.)
5. Check the compatibility table in “Concurrency and Compatibility” on page 2-160 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 2-160.

Invoking QUIESCE

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for an explanation of ways to invoke DB2 utilities.

Phases and Data Flow

The QUIESCE utility operates in these phases:

Phase	Description
UTILINIT	Initialization and setup
QUIESCE	Determining the quiesce point and updating the catalog
UTILTERM	Cleanup

Creating JCL Statements for Required Data Sets

Table 30 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 30. 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 objects are named in the utility control statement and do not require DD cards 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 2-161 for QUIESCE syntax and option descriptions. See “Sample JCL and Control Statements” on page 2-162 for examples of QUIESCE usage.

Before Running QUIESCE

You cannot run QUIESCE on a table space that is in copy pending, check pending, or recovery pending status. See “Resetting the Copy Pending Status” on page 2-103, “Resetting Check Pending Status” on page 2-42, and “Resetting the Recovery Pending Status” on page 2-104 for information about resetting these statuses.

Instructions for Specific Tasks

For the tasks described, specify the options and values shown with your utility control statement. The following tasks are described here:

Using QUIESCE for Recovery

You can recover a table space to its quiesce point with the RECOVER TABLESPACE utility. See “Chapter 2-13. RECOVER TABLESPACE” on page 2-173 for information about the RECOVER TABLESPACE utility.

Using QUIESCE to Automate Copy

You can use the QUIESCE utility to determine whether table spaces are in check pending, copy pending, or recovery pending status, and whether any I/O errors occur while accessing the table space data sets. A successful execution of QUIESCE utility usually indicates that you can create valid image copies of those table spaces. You can use the return code from QUIESCE utility to decide whether to execute the COPY utility step.

Obtaining a Common QUIESCE Point

You can use the QUIESCE utility to obtain a common quiesce point for a set of referentially-related table spaces. First, use the REPORT TABLESPACESET utility to obtain the list of table spaces in the set. Next, run the QUIESCE utility specifying the entire set of table spaces in a list. If you 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.

Considerations for Running QUIESCE

If a Table Space is in a Pending Status: If QUIESCE is run on a table space in "copy pending," "check pending", or "recovery pending" pending status, it terminates as shown in Figure 19.

```
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 19. 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.

| ***If Too Many Table Spaces are Specified:*** If QUIESCE is run on a list of more
| than 1165 table spaces, it terminates with a return code of 8 as shown below:

```
DSNU000I   DSNUGUTC - OUTPUT START FOR UTILITY,
                      UTILID = QUIESCE
DSNU046I   DSNUGPSP - UTILITY STATEMENT IS TOO LONG OR TOO
                      COMPLEX
DSNU012I   DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST
                      RETURN CODE=8
```

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 2-28.

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 type 1 nonpartitioned index exists on a partitioned table space, utilities operating on different partitions of a table space can be incompatible because of contention on the nonpartitioned index.

Table 31 shows which claim classes QUIESCE drains and any restrictive state the utility sets on the target object.

Table 32 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 31. QUIESCE. Use of claims and drains; restrictive states set.

Target	WRITE YES	WRITE NO
Table space or partition	DW/UTRO	DW/UTRO
Index or partition	DW/UTRO	
Nonpartitioned 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 32 (Page 1 of 2). QUIESCE Compatibility

Action	QUIESCE
CHECK DATA	No
CHECK INDEX	Yes
COPY SHRLEVEL REFERENCE	Yes

Table 32 (Page 2 of 2). QUIESCE Compatibility

Action	QUIESCE
COPY SHRLEVEL CHANGE	No
DIAGNOSE	Yes
LOAD	No
MERGECOPY	Yes
MODIFY	Yes
QUIESCE	Yes
RECOVER	No
REORG INDEX	No
REORG TABLESPACE UNLOAD CONTINUE or PAUSE	No
REORG TABLESPACE UNLOAD ONLY	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.

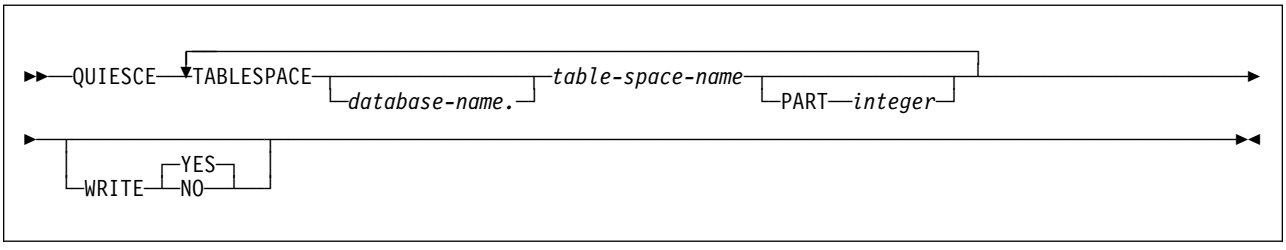
| QUIESCE on SYSUTILX is an “exclusive” job; such a job can interrupt another job
| between job steps, possibly causing the interrupted job to time out.

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 1-3.



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 2-7.

TABLESPACE *database-name.table-space-name*
Specifies the table space to be quiesced.

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

table-space-name
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.

WRITE
Specifies whether to write the changed pages from the table spaces to DASD.

YES
Establishes a quiesce point and writes the changed pages from the table spaces to DASD.

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

NO
Establishes a quiesce point but does not write the changed pages from the table spaces to DASD.

Sample JCL and Control Statements

Examples

Example 1: Sample JCL for QUIESCE: Establish a quiesce point for three table spaces.

```
//UTILSAMP EXEC PGM=DSNUTILB,REGION=1024K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*
//*********************************************************************
//*  DATA SETS USED BY THE UTILITY                                    *
//*********************************************************************
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
  QUIESCE TABLESPACE DSN8D51A.DSN8S51D
          TABLESPACE DSN8D51A.DSN8S51E
          TABLESPACE DSN8D51A.DSN8S51P
//*
```

Example 2: Sample Control Statement for QUIESCE: Establish a quiesce point for the DSN8D51A.DSN8S51E and DSN8D51A.DSN8S51D table spaces.

```
QUIESCE TABLESPACE DSN8D51A.DSN8S51E TABLESPACE DSN8D51A.DSN8S51D
```

The following is output of the preceding command:

```
  DSNU000I   DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TSLQ.STEP1
  DSNU050I   DSNUGUTC - QUIESCE TABLESPACE DSN8D51A.DSN8S51E
                                TABLESPACE DSN8D51A.DSN8S51D
  DSNU477I . DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D51A.DSN8S51E
  DSNU477I . DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D51A.DSN8S51D
  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 TABLESPACE    DSN8D51A.DSN8S51D
        TABLESPACE    DSN8D51A.DSN8S51E
        TABLESPACE    DSN8D51A.PROJ
        TABLESPACE    DSN8D51A.ACT
        TABLESPACE    DSN8D51A.PROJACT
        TABLESPACE    DSN8D51A.EMPPROJA
```

The following is output of the preceding command:

```
  DSNU000I   DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TSLQ.STEP1
  DSNU050I   DSNUGUTC - QUIESCE TABLESPACE DSN8D51A.DSN8S51D
                                TABLESPACE DSN8D51A.DSN8S51E
                                TABLESPACE DSN8D51A.PROJ
                                TABLESPACE DSN8D51A.ACT
                                TABLESPACE DSN8D51A.PROJACT
                                TABLESPACE DSN8D51A.EMPPROJA
  DSNU477I . DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D51A.DSN8S51D
  DSNU477I . DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D51A.DSN8S51E
  DSNU477I . DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D51A.PROJ
  DSNU477I . DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D51A.ACT
  DSNU477I . DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D51A.PROJACT
  DSNU477I . DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSN8D51A.EMPPROJA
  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
```

Chapter 2-12. RECOVER INDEX

RECOVER INDEX recreates indexes from the table that they reference.

Syntax Diagram: For a diagram of RECOVER INDEX syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 2-170.

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 SYSADM or installation SYSOPR authority can also execute RECOVER TABLESPACE, but only on a table space in the DSNDB01 or DSNDB06 database.

Instructions for Running RECOVER INDEX

In order to run RECOVER INDEX, you must:

1. Read “Before Running RECOVER INDEX” on page 2-167 in this chapter.
2. Prepare the necessary data sets, as described in “Creating JCL Statements for Required Data Sets” on page 2-166.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for RECOVER INDEX, see “Sample JCL and Control Statements” on page 2-172.)
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 2-167. (For a complete description of the syntax and options for RECOVER INDEX, see “Syntax and Options of the Control Statement” on page 2-170.)
5. Check the compatibility table in “Concurrency and Compatibility” on page 2-168 if you want to run other jobs concurrently on the same target objects.
6. Plan for restart if the RECOVER INDEX job doesn't complete, as described in “Terminating or Restarting RECOVER INDEX” on page 2-168.

Invoking RECOVER INDEX

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for an explanation of ways to invoke DB2 utilities.

Phases and Data Flow

The RECOVER 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
UTILTERM	Cleanup

Creating JCL Statements for Required Data Sets

Table 33 describes the data sets used by RECOVER INDEX. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 33. Data Sets Used by RECOVER 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	Temporary data set used to store the index keys for RECOVER INDEX. Its DD name is specified by 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 set, see page 2-166.	No
Input data sets	Image copy data sets which are preallocated. Their DD names are specified by the user.	No

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

Table space
Object to be recovered. It is named in the RECOVER INDEX control statement and is accessed through the DB2 catalog.

Creating the Work Data Set: RECOVER 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 recovered on the table.
2. Add the products obtained in step 1.
3. Multiply the sum (from step 2) by the largest key length plus 13.

Allocating twice the space used by the input data sets is usually adequate for the sort work data sets. One or two 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 2-170 for syntax and option descriptions. See “Sample JCL and Control Statements” on page 2-172 for examples of usage.

Before Running RECOVER INDEX

Because the data needed to recover an index is in the table space on which the index is based, you do not need image copies of indexes. To recover the index, you do not need to recover the table space, unless it also is damaged. Neither do you have to recover 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, you must recover all of the indexes. RECOVER TABLESPACE does not recover index spaces. If you need to recover both a table space and one or more of its index spaces, recover the table space first. RECOVER INDEX recovers indexes by re-creating them from the tables on which they are based.

Instructions for Specific Tasks

For the tasks described, specify the options and values shown with your utility control statement. The following tasks are described here:

Recovering Index Partitions

RECOVER INDEX can recover one or more partitions of a partitioned index. The PART option allows you to specify a particular partition to be recovered. This prevents RECOVER INDEX from unnecessarily scanning the entire table space, and unnecessarily recovering every index. If you recover any part of a partitioned table space to the latest point of consistency, you must recover all nonpartitioned indexes.

Improving Performance and Space Utilization

RECOVER INDEX recovers indexes by re-creating them from the tables on which they are based. It can recover one or more partitions of a partitioned index in a partitioned table space. The PART option allows you to specify a particular partition to be recovered. This prevents RECOVER INDEX from unnecessarily scanning the entire table space and recovering every index.

To recover several indexes simultaneously and reduce recovery time, submit multiple RECOVER INDEX jobs on indexes in the same table space. Each job accesses the table space concurrently during key extraction. After the keys are extracted, each index recovery job independently builds an index.

When recovering nonpartitioned indexes and partitions of partitioned indexes, this type of parallel processing on the same table space decreases the size of the sort data set, as well as the total time required to sort all the keys.

When you run the RECOVER INDEX utility concurrently on separate partitions of a partitioned index, the sum of the processor time will be roughly equivalent to the time it takes to run a single RECOVER INDEX job against the entire index. For partitioned indexes, the elapsed time for running concurrent RECOVER INDEX jobs will be a fraction of the elapsed time for running a single RECOVER INDEX job against an entire index.

RECOVER 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.

Resetting Recovery Pending Status

You can reset the recovery pending status for an index with any of these operations:

- RECOVER INDEX
- REPAIR with SET NORCVRPEND
- -START DATABASE with ACCESS FORCE.

Recovering Critical Catalog Indexes

An ID with a granted authority receives message DSNT500I, “RESOURCE UNAVAILABLE,” while trying to recover 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.

Terminating or Restarting RECOVER INDEX

You can terminate RECOVER INDEX with the TERM UTILITY command. If RECOVER is terminated, the table space is placed in the "recover pending" status and is unavailable until it has been successfully recovered.

If you use WORKDDN, RECOVER INDEX can be restarted during the UNLOAD and SORT phases, and at the last index built during the BUILD phase. However, there is a small window 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 the SORT phase. If you omit WORKDDN, the job starts over again from the beginning.

For more guidance in restarting online utilities, see “Restarting an Online Utility” on page 2-28.

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 type 1 nonpartitioned index exists on a partitioned table space, utilities operating on different partitions of a table space can be incompatible because of contention on the nonpartitioned index.

Table 34 on page 2-169 shows which claim classes RECOVER INDEX drains and any restrictive state the utility sets on the target object.

Table 35 on page 2-169 shows which utilities can run concurrently with RECOVER 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 34. RECOVER INDEX. Use of claims and drains; restrictive states set.

Target	RECOVER INDEX	RECOVER INDEX PART
Table space or partition	DW/UTRO	DW/UTRO
Index or physical partition	DA/UTUT	DA/UTUT
Nonpartitioned type 1 index	DA/UTUT	DA/UTUT
Nonpartitioned type 2 index	DA/UTUT	DR
Logical partition of type 2 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.

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

Table 35 (Page 1 of 2). RECOVER INDEX Compatibility

Action	RECOVER INDEX
CHECK DATA	No
CHECK INDEX	No
COPY SHRLEVEL REFERENCE	Yes
COPY SHRLEVEL CHANGE	No
DIAGNOSE	Yes
LOAD	No
MERGECOPY	Yes
MODIFY	Yes
QUIESCE	No
RECOVER INDEX	No
RECOVER TABLESPACE	No
REORG INDEX	No
REORG UNLOAD CONTINUE or PAUSE	No
REORG UNLOAD ONLY without cluster index	Yes
REORG UNLOAD ONLY 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

Table 35 (Page 2 of 2). RECOVER INDEX Compatibility

Action	RECOVER INDEX
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 TABLESPACE	Yes
RUNSTATS INDEX	No
STOSPACE	Yes

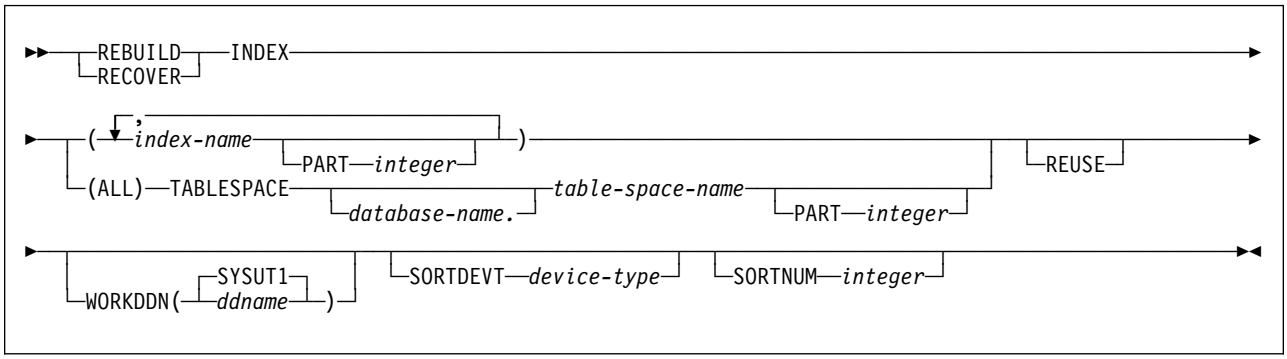
To run on SYSIBM.DSNLUX01 or SYSIBM.DSNLUX02, RECOVER INDEX must be the only utility in the job step and the only utility running in the DB2 subsystem.

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 1-3.



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 2-7.

Notice of Future Change

Only the REBUILD INDEX syntax will be supported in the future. Use of RECOVER INDEX should be replaced by REBUILD INDEX.

(*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 partitioned index or the logical partition of a nonpartitioned 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 RECOVER INDEX 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 recovering the index because of a media failure, do not specify REUSE.

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 param-

eter 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.

Sample JCL and Control Statements

Examples

Example 1: Recover an Index: Recover the DSN8510.XDEPT1 index, which indexes the DSN8510.TDEPT table in the DSN8D51A database.

```
//UTILSAMP EXEC PGM=DSNUTILB,REGION=1024K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*
//*********************************************************************
//*  DATA SETS USED BY THE UTILITY                                    *
//*********************************************************************
//SORTWK01 DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSPRINT DD  SYSOUT=*
//UTPRINT  DD  SYSOUT=*
//SYSIN    DD  *
    RECOVER INDEX (DSN8510.XDEPT1)
//*
```

Example 2: Recover Index Partitions: Recover partitions 2 and 3 of the DSN8510.XEMP1 index.

```
RECOVER INDEX (DSN8510.XEMP1 PART 2, DSN8510.XEMP1 PART 3)
```

Chapter 2-13. RECOVER TABLESPACE

The RECOVER TABLESPACE online utility recovers data to the current state or to a previous point in time.

The largest unit of data recovery is the table space; the smallest is the page. RECOVER TABLESPACE recovers an entire table space, a partition or data set, pages within an error range, or a single page. You recover data from image copies of a table space and from log records containing changes to the table space. 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 TABLESPACE uses the previous image copy data sets. RECOVER TABLESPACE does not recover indexes.

Syntax Diagram: For a diagram of RECOVER TABLESPACE syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 2-190.

Output: Output from RECOVER TABLESPACE consists of recovered data (either a table space, 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 TABLESPACE puts any associated index spaces in recovery pending status. You must run RECOVER INDEX to remove the index space from recover pending status.

If you use RECOVER TABLESPACE to recover a set of referentially related table spaces, then you must ensure that you recover the entire set of table spaces to a common quiesce point. 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, then RECOVER TABLESPACE puts all dependent table spaces of the inconsistent table spaces in the set in 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 TABLESPACE, but only on a table space in the DSNDB01 or DSNDB06 database.

Instructions for Running RECOVER TABLESPACE

In order to run RECOVER TABLESPACE, you must:

1. Read “Before Running RECOVER TABLESPACE” on page 2-175 in this chapter.
2. Prepare the necessary data sets, as described in “Creating JCL Statements for Required Data Sets” on page 2-174.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for

RECOVER TABLESPACE, see "Sample JCL and Control Statements" on page 2-195.)

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 2-175. (For a complete description of the syntax and options for RECOVER TABLESPACE, see "Syntax and Options of the Control Statement" on page 2-190.)
5. Check the compatibility table in "Concurrency and Compatibility" on page 2-188 if you want to run other jobs concurrently on the same target objects.
6. Plan for restart if the RECOVER TABLESPACE job does not complete, as described in "Terminating or Restarting RECOVER TABLESPACE" on page 2-188.

Invoking RECOVER TABLESPACE

See "Chapter 2-1. Invoking DB2 Online Utilities" on page 2-7 for an explanation of ways to invoke DB2 utilities.

Phases and Data Flow

The RECOVER TABLESPACE 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
LOGAPPLY	Apply any outstanding log changes to the table space restored from the previous phase or step
UTILTERM	Cleanup

Creating JCL Statements for Required Data Sets

Table 36 describes the data sets used by RECOVER TABLESPACE. Include statements in your JCL for each required data set, and any optional data sets you want to use.

Table 36. Data Sets Used by RECOVER TABLESPACE

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 The name of the table 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 want to recover to a particular backup copy, without any of the updates applied, you can use the TOCOPY option and specify the name of the image copy data set.

When RECOVER TABLESPACE is used, RECOVER refers to the SYSIBM.SYSCOPY catalog table. 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.

Before Running RECOVER TABLESPACE

Recovering Data and Indexes: You do not always need to recover both the data and indexes. Only if the indexes are damaged or the table space is recovered to a point in time do you need to recover the indexes. However, if you need to recover both data and indexes, then the data must be recovered first. The data is recovered from the image copies and logs. The indexes are then recovered by the RECOVER INDEX utility from the data.

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 is located, RECOVER places the data set on another volume of the storage group.

Instructions for Specific Tasks

For the tasks described, specify the options and values shown with your utility control statement. The following tasks are described here:

Recovering a Table Space

The following RECOVER statement recovers table space DSN8S51D in database DSN8D51A:

```
RECOVER TABLESPACE DSN8D51A.DSN8S51D
```

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 DSN8D51A.DSN8S51E, and recovers the table space DSN8D51A.DSN8S51D to the quiesce point (RBA X'000007425468').

```
RECOVER TABLESPACE DSN8D51A.DSN8S51E DSNUM 2
        TABLESPACE DSN8D51A.DSN8S51D
        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 DSN8D51A, table space DSN8S51E:

```
RECOVER TABLESPACE DSN8D51A.DSN8S51E DSNUM 1
        TABLESPACE DSN8D51A.DSN8S51E DSNUM 2
        TABLESPACE DSN8D51A.DSN8S51E DSNUM 3
        TABLESPACE DSN8D51A.DSN8S51E 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 2-103.

Recovering a Table Space List

You can recover a list of table spaces. 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.

The merging of incremental copies is done serially and dynamically. Because of this, 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.

Recovering a Data Set or Partition

You can use the RECOVER TABLESPACE 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
```

On the other hand, if image copies are taken at the table space level, individual data sets can be recovered simply by coding the DSNUM parameter.

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 could 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 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 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 TABLESPACE 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.

Recovering a Page

RECOVER TABLESPACE 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 TABLESPACE 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 TABLESPACE 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 TABLESPACE utility job specifying TABLESPACE(TSPACE1) PAGE(*5*) CONTINUE. The RECOVER TABLESPACE 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 TABLESPACE 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 TABLESPACE:

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 TABLESPACE, you should run the ICKDSF service utility to correct the DASD error. If an I/O error is detected during the RECOVER TABLESPACE process, message DSNU538I is issued 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, 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. If you then continue to recover to the current status, you do not want RECOVER to begin by restoring the data set from a DB2 image copy. Use the LOGONLY option of 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.

The LOGONLY option can be used on single table spaces, table space lists, and partitions. It also supports both recover to a current point in time and recover TORBA (to a prior point in time).

To obtain an 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*)

To ensure that no other transactions can access DB2 objects between the time that a data set is restored and the time that RECOVER LOGONLY is run:

1. Stop the DB2 objects being recovered by issuing the following command:

   -STOP DATABASE(*database name*) SPACENAM(*tablespace-name*)
2. Start the DB2 objects being recovered by issuing the following command:

   -START DATABASE(*database name*) SPACENAM(*tablespace-name*) ACCESS(UT)
3. Restore all DB2 data sets that are being recovered.
4. Run the RECOVER TABLESPACE utility without the TORBA parameter 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 TABLESPACE utility with both the TORBA and LOGONLY parameters.
5. If you recovered the DB2 objects to a prior point in time, recover 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(*tablespace-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. DSNDB01.DBD01

3. All indexes on SYSUTILX
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, SYSPLAN, SYSPKAGE, SYSSTATS, SYSSTR, and SYSVIEWS. Most indexes are listed in *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.

Recovery of the items on the list can be done in parallel 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 TABLESPACE statement appears must not contain any other utility statements. No other utilities can run while the RECOVER utility is running.
   - DSNDB01.SYSUTILX
   - DSNDB01.DBD01
   - All indexes on SYSUTILX
2. When you recover the following table spaces, no other utilities can run while the RECOVER TABLESPACE 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 *Administration Guide*.

---

[3] If there are no user defined indexes on the catalog, execute `RECOVER INDEX (ALL) TABLESPACE DSNDB06.SYSxxxx` to recover 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 recovered individually and the user defined indexes recovered in a later step. See Appendix D of *SQL Reference* for a list of the IBM defined indexes.

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: In order to recover one object, RECOVER must obtain information about it from some other object. Table 37 lists the objects from which RECOVER must obtain information.

Table 37. Table Spaces 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. Recovery for the SYSUTILX indexes DSNLUX01 and DSNLUX02 is performed by the RECOVER INDEX utility, which is not restartable.
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 of the first log record after the most recent copy.
DSNDB06.SYSDBAUT, DSNADX01, DSNDB06.SYSUSER	To verify that the authorization ID is authorized to run RECOVER.
DSNDB06.SYSDBASE, DSNDSX01	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 the sample library that can be used to check the consistency of the catalog.

Indexes are re-created by RECOVER INDEX. If you have recovered table spaces in the catalog or directory, you could have to re-create their indexes. Use the CHECK INDEX utility to determine whether an index is inconsistent with the data it indexes.

You must recover the catalog and directory before recovering user table spaces. Database DSNDB06 must be started before running the RECOVER TABLESPACE utility on any objects outside of the catalog and directory.

DSNDB01.SYSUTILX, DSNDB01.DBD01, DSNDB01.SYSLGRNX, DSNDB06.SYSCOPY, DSNDB06.SYSGROUP, DSNDB01.SCT02, and DSNDB01.SPT01 do not have entries in SYSIBM.SYSLGRNX. They are assumed to be open from the point of their last image copy, so the RECOVER TABLESPACE utility processes the log from that point onward.

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 TABLESPACE utility on the catalog or directory using an authorization ID with the installation SYSADM or installation SYSOPR authority.

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. 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 2-186. DB2 records the RBAs 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 leaves data in a consistent state and indexes in an inconsistent state, all indexes must be recovered by running RECOVER INDEX.

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, 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 later RECOVER TOCOPY operation can produce inconsistent data.

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 or on a single data set of a simple table space. In either case, all data sets must be restored to the same level or the data will be inconsistent.

Point-in-time recovery does not restore related indexes to a consistent point, as does recovery to the current point in time, when the indexes are already consistent. Therefore, after a recovery made with TOCOPY or TORBA, or TOLOGPOINT, indexes are placed in recovery pending status. See "Resetting Recovery Pending Status" on page 2-186 for information about resetting this status.

Point-in-time recovery can cause table spaces to be placed in *check pending* status if they have table check constraints or referential constraints defined on them. When recovering tables 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 resetting the check pending status, see “Chapter 2-4. CHECK DATA” on page 2-37.

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 *Administration Guide*.

Avoiding Specific Image Copy Datasets

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    DSNUBCR3 - UNABLE TO ALLOCATE R92341Q.UTQPS001.FCOPY010
                       RC=4, CODE=X'04840000'
DSNU508I    DSNUBCR2 - 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    DSNUBCR3 - UNABLE TO ALLOCATE R92341Q.UTQPS001.FCOPY010,
                       RC=4, CODE=X'17080000'
DSNU508I    DSNUBCR2 - 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 image copies of the table space before recovering the table space. If you do not merge your image copies then RECOVER TABLESPACE merges them. If when merging the image copies, RECOVER TABLESPACE cannot allocate all the incremental image copy data sets, then RECOVER TABLESPACE uses the log instead.

Include a table space list on RECOVER TABLESPACE to avoid scanning the log more than once.

If you use RECOVER TABLESPACE TOCOPY for full image copies, you can improve performance by using data compression. The improvement is proportional to the degree of compression.

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.

- 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 later 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.

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 know 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.

The maximum number of input tape units that are used to read the archive log can be dynamically changed with 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 jobs 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 TABLESPACE job successfully allocates all available units, the job waits for more units to become available.

Resetting Recovery Pending Status

Several possible operations on a table space or index space place it in the *recovery pending* status. It can be turned off in several ways, listed below:

- Recover the table space, index space, or partition.
- 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.
- Start the database that contains the table space or index space with ACCESS(FORCE).

The last two operations, REPAIR and -START DATABASE, do not fix the data inconsistency in the table space or index.

Considerations for Running RECOVER TABLESPACE

This section includes additional information to keep in mind when running RECOVER TABLESPACE.

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, but not including, the missing incremental image copy
- Not use the remaining incremental image copy data sets.

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 TABLESPACE 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 TABLESPACE utility terminates.

If you always make multiple image copies, RECOVER TABLESPACE should seldom fall back to an earlier point. Instead, RECOVER TABLESPACE relies upon the backup copied data set should the primary copied data set not be usable.

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 could 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 TABLESPACE utility can re-access the damaged media. You must:

1. Use -ALTER STOGROUP to remove the bad volume and add another
2. Invoke the RECOVER TABLESPACE utility for all table spaces on that volume.

If the RECOVER TABLESPACE 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 TABLESPACE utility automatically redefines the cluster. For user-defined table spaces or indexes, you must redefine the cluster before invoking the RECOVER TABLESPACE utility.

Recovering Table Spaces with Mixed Volume IDs

You cannot run RECOVER TABLESPACE on a table space on which mixed specific and non-specific volume IDs were defined with CREATE STOGROUP or ALTER STOGROUP.

Terminating or Restarting RECOVER TABLESPACE

For instructions on restarting a utility job, see “Restarting an Online Utility” on page 2-28.

Terminating RECOVER TABLESPACE: Terminating a RECOVER TABLESPACE job with the -TERM UTILITY command leaves the object being recovered in recovery pending state. The data is unavailable until the object has been successfully recovered.

Restarting RECOVER TABLESPACE: RECOVER TABLESPACE can be restarted at the beginning of the phase or at the current checkpoint.

If you attempt to recover multiple table spaces using a single RECOVER statement and the utility fails in:

- The RESTORE phase: All successfully restored table spaces are placed in the recovery pending status and the status of the remaining table spaces is unchanged.
- The LOGAPPLY phase: All table spaces specified in the RECOVER TABLESPACE statement are placed in the recovery pending status.

In both cases, the causes of the failure need to be identified and fixed before a current restart is performed.

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 type 1 nonpartitioned index exists on a partitioned table space, utilities operating on different partitions of a table space can be incompatible because of contention on the nonpartitioned index.

Table 38 shows which claim classes RECOVER TABLESPACE claims and drains and any restrictive state the utility sets on the target object.

Table 39 on page 2-189 shows which utilities can run concurrently with RECOVER TABLESPACE 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 38 (Page 1 of 2). RECOVER TABLESPACE. Use of claims and drains; restrictive states set.

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*

Table 38 (Page 2 of 2). RECOVER TABLESPACE. Use of claims and drains; restrictive states set.

Target	RECOVER (no option)	RECOVER TORBA or TOCOPY	RECOVER PART TORBA or TOCOPY	RECOVER ERROR-RANGE
Index or physical partition		DA/UTUT	DA/UTUT	
Nonpartitioned type 1 index		DA/UTUT	DA/UTUT	
Nonpartitioned type 2 index		DA/UTUT	DR	
Logical partition of type 2 index			DA/UTUT	
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

* During the UTILINIT phase, the claim and restrictive state change from DA/UTUT to CW/UTRW.

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

Table 39 (Page 1 of 2). RECOVER TABLESPACE Compatibility

Action	RECOVER (no option)	RECOVER TOCOPY or TORBA	RECOVER ERROR-RANGE
CHECK	No	No	No
COPY	No	No	No
DIAGNOSE	Yes	Yes	Yes
LOAD	No	No	No
MERGECOPY	No	No	No
MODIFY RECOVERY	No	No	No
MODIFY STATISTICS	Yes	Yes	Yes
QUIESCE	No	No	No
RECOVER	No	No	No
REORG INDEX	Yes	No	Yes
REORG TABLESPACE	No	No	No

Table 39 (Page 2 of 2). RECOVER TABLESPACE Compatibility

Action	RECOVER (no option)	RECOVER TOCOPY or TORBA	RECOVER ERROR-RANGE
REPAIR LOCATE TABLESPACE	No	No	No
REPAIR LOCATE INDEX	Yes	No	Yes
REPORT	Yes	Yes	Yes
RUNSTATS INDEX	Yes	No	Yes
RUNSTATS TABLESPACE	No	No	No
STOSPACE	Yes	Yes	Yes

To run on DSNDB01.SYSUTILX, RECOVER TABLESPACE must be the only utility in the job step and the only utility running in the DB2 subsystem.

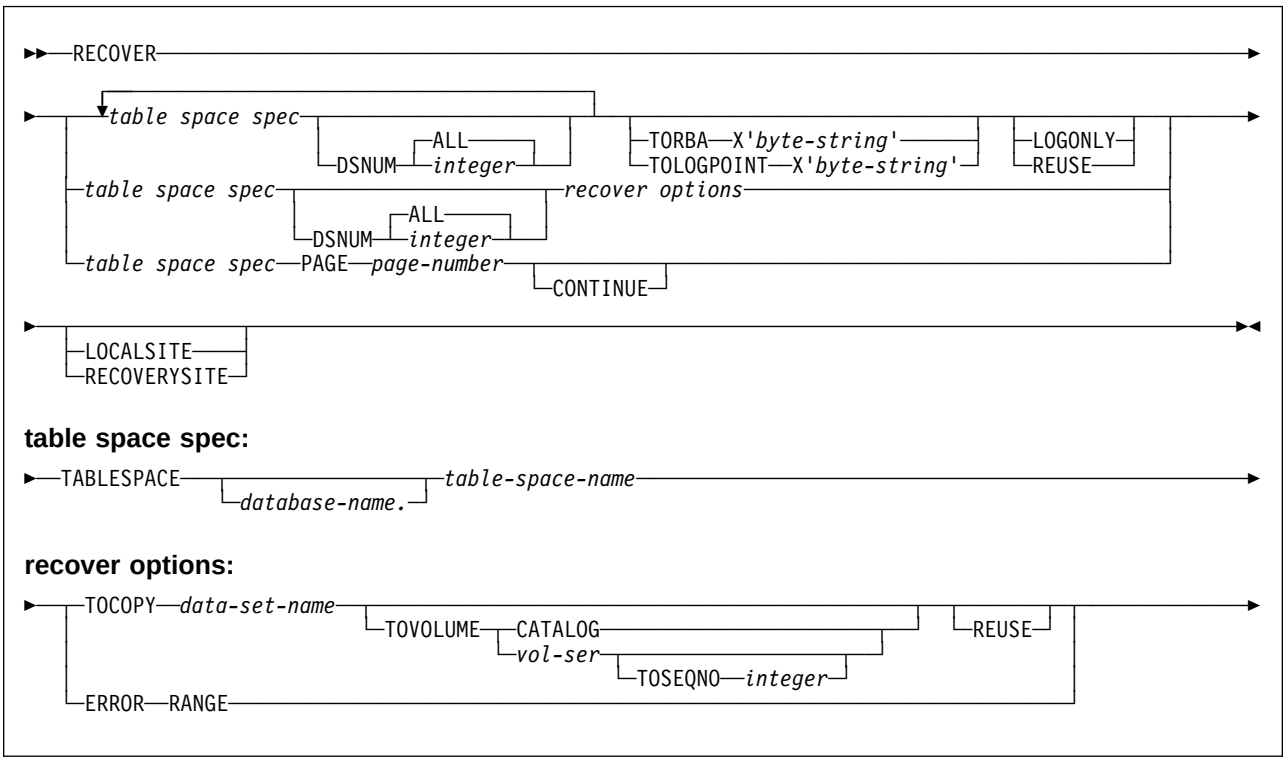
RECOVER TABLESPACE on any catalog or directory table space is an “exclusive” job; such a job can interrupt another job between job steps, possibly causing the interrupted job to time out.

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 1-3.



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 2-7.

TABLESPACE *database-name.table-space-name*
Specifies the table space (and optionally, the data base to which it belongs) that is to be recovered.

RECOVER cannot recover a table 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, the job terminates and you receive error message DSNU419I.

You can specify a list of table spaces by repeating the TABLESPACE keyword. If you use a list of table spaces, the valid keywords are: DSNUM, TORBA, TOLOGPOINT, LOGONLY, and LOCALSITE/RECOVERYSITE. You cannot recover 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.

The following are optional.

DSNUM
Identifies a partition or data set, within the table space, that is to be recovered; or it recovers the entire table space.

ALL
Recovers the entire table space.

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

integer
Is the number of a partition or data set to be recovered. 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*n*

where:

catname The VSAM catalog name or alias
x C or D
dbname The database name
tsname The table space name
n 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 TABLESPACE 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. 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 digits. If *byte-string* is the RBA of the first byte of a log record, that record is included in the recovery.

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 TABLESPACE 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.

This option resets the check pending status when:

- All members of a table space set are recovered to the same quiesce point, and no referential constraints were defined on a dependent table after that

quiesce point. The check pending status is reset for any table space in the table space set.

This option sets the check pending status when:

- One or more members of a table space set are recovered to a different quiesce point, so that all members of the table space set are not recovered to the same point in time. Dependent table spaces that are recovered (and their 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.

TOLOGPOINT X'*byte-string***'**
Is used to specify a point on the log to recover to. Specify either an RBA or an LRSN value.

The LRSN is a string of 12 hexadecimal digits 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 TABLESPACE 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 a table space because of a media failure, do not specify
REUSE.

LOCALSITE
Causes RECOVER TABLESPACE to use image copies from the local site. If neither LOCALSITE or RECOVERYSITE is specified, then image copies from the current site of invocation are used. (The current site is identified on the installation panel DSNTIPO under SITE TYPE and in the macro DSN6SPRM under SITETYP.)

RECOVERYSITE
Causes RECOVER TABLESPACE to use image copies from the recovery site. If neither LOCALSITE or RECOVERYSITE is specified, then image copies from the current site of invocation are used. (The current site is identified on the installation panel DSNTIPO under SITE TYPE and in the macro DSN6SPRM under SITETYP.)

TOCOPY *data-set-name*
Terminates the recovery process after applying a particular image copy data set.

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. The effect of using TOCOPY is the same as using TORBA or TOLOGPOINT and naming one of the following:

- The RBA associated with a SHRLEVEL CHANGE image copy plus 1
- The RBA associated with a SHRLEVEL REFERENCE image copy.

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 TABLESPACE 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. 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.)

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 TABLESPACE utility will run without the ERROR RANGE option.

Refer to Section 4 (Volume 1) of *Administration Guide* for additional information concerning the use of this keyword.

Sample JCL and Control Statements

Examples

Example 1: Recover an Error Range: Recover from reported media failure in partition 2 of table space DSN8D51A.DSN8S51D.

```
//UTILSAMP EXEC PGM=DSNUTILB,REGION=4096K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*
//*********************************************************************
//*  DATA SETS USED BY THE UTILITY                                    *
//*********************************************************************
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
    RECOVER TABLESPACE DSN8D51A.DSN8S51D DSNUM 2 ERROR RANGE
/*
```

Example 2: RECOVER a Table Space: Recover table space DSN8S51D, in database DSN8D51A.

```
RECOVER TABLESPACE DSN8D51A.DSN8S51D
```

Example 3: RECOVER a Table Space Partition: Recover the second partition of table space DSN8S51D.

```
RECOVER TABLESPACE DSN8D51A.DSN8S51D DSNUM 2
```

Example 4: RECOVER a Table Space to a Specific RBA: Recover table spaces DSN8D51A.DSN8S51E and DSN8D51A.DSN8S51D to their quiesce point (RBA X'000007425468').

```
RECOVER TABLESPACE DSN8D51A.DSN8S51E DSNUM 2
     TABLESPACE DSN8D51A.DSN8S51D
     TORBA (X'000007425468')
```

Chapter 2-14. REORG

The REORG online utility reorganizes a table space or index to improve access performance and reclaim fragmented space. In addition, the utility can reorganize a single partition of either a partitioned index or a partitioned table space. You can specify the degree of access to your data during reorganization. If you specify REORG UNLOAD ONLY, the data is unloaded in a format that is acceptable to the LOAD utility of the same DB2 subsystem.

Syntax Diagram: For a diagram of REORG syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 2-225.

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.

Authorization Required: To execute this utility on a user table space or 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 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 2-201).

An authority other than installation SYSADM or installation SYSOPR can receive message DSNT500I, “resource unavailable,” while trying to reorganize a tablespace 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.

An ID with installation SYSOPR authority can also execute REORG INDEX, but only on an index in the DSNDB06 database.

Instructions for Running REORG

In order to run REORG, you must:

1. Read “Before Running REORG” on page 2-201 in this chapter.
2. Prepare the necessary data sets, as described in “Creating JCL Statements for Required Data Sets” on page 2-199.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for REORG, see “Sample JCL and Control Statements” on page 2-245.)
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 2-204. (For a complete description of the syntax and options for REORG, see “Syntax and Options of the Control Statement” on page 2-225.)
5. Check the compatibility table in “Concurrency and Compatibility” on page 2-219 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” on page 2-216.

Invoking REORG

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for an explanation of ways to invoke DB2 utilities.

Phases and Data Flow

The REORG utility operates in these phases:

Phase	Description
UTILINIT	Initialization and setup
UNLOAD	Unloads table space; sorts data if a clustering index exists and either SORTDATA or SHRLEVEL CHANGE is specified. If NOSYSREC is specified, 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 COPYDDN, RECOVERYDDN, SHRLEVEL REFERENCE, or SHRLEVEL CHANGE are specified. If SORTKEYS is used, a subtask sorts the index keys.
SORT	Sorts index keys. If SORTKEYS is used, passes sorted keys in memory to the BUILD phase.
BUILD	Builds indexes; corrects nonpartitioned indexes if REORG TABLESPACE PART SHRLEVEL NONE is specified.
LOG	Processes log iteratively; appends changed pages to the full image copies. Used only if SHRLEVEL CHANGE is specified.
SWITCH	Switches access to shadow copy of table space or partition. Used only if SHRLEVEL REFERENCE or CHANGE is specified.
BUILD2	Corrects nonpartitioned indexes if REORG TABLESPACE PART SHRLEVEL REFERENCE or CHANGE is specified.
UTILTERM	Cleanup

Creating JCL Statements for Required Data Sets

Table 40 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 40. Data Sets Used by REORG

Data Set	Description	Required?
SYSIN	Input data set containing the utility control statement.	Yes
SYSPRINT	Output data set for messages.	Yes
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]
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.

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 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)
   ```

3. If you have variable length fields, the calculation in 2 could 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 numrows = the number of rows in the data set
        longrow = length of the longest row in the table space

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 2-209 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 use 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 needed 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, multiply the number of records in the table by the number of indexes defined on the table.
2. Add the products obtained in step 1.
3. Multiply that sum by the largest key length plus 13.

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 ONLY
- UNLOAD PAUSE without KEEPDICTIONARY
- UNLOAD CONTINUE without KEEPDICTIONARY

One or two large SORTWK*nn* data sets are preferable to several small ones. If adequate space is not available, REORG cannot be run.

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 is:

```
//UTPRINT DD  SYSOUT=A
```

Creating the Control Statement

See “Syntax and Options of the Control Statement” on page 2-225 for REORG syntax and option descriptions. See “Sample JCL and Control Statements” on page 2-245 for examples of REORG usage.

Before Running REORG

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 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 TYPE 2 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.

If you attempt concurrent invocations of REORG on several table spaces and/or several partitions of a table space, and the invocations use the same mapping table, then the invocations will execute serially. If the invocations use different mapping tables, then they can execute concurrently. The mapping tables need not reside in separate table spaces.

For a sample of REORG with SHRLEVEL CHANGE and a sample mapping table and index, see job sample DSNTEJ1 as described in *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 *Administration Guide*. The data sets must already exist when you invoke REORG. The names have the form `catname.DSNDBx.dbname.psname.S0001.Annn`. Define the data sets as LINEAR and use SHAREOPTIONS(3,3). However, if the data base is ROSHARE OWNER, use SHAREOPTIONS(1,3), as described in Appendix F (Volume 2) of *Administration Guide*. When reorganizing an entire table space, you must create the shadow data sets for the table space and all indexes. 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 with SHRLEVEL REFERENCE or SHRLEVEL CHANGE on partition *mmm* of a partitioned table space, you must create, for each nonpartitioned index that resides in user defined data sets, a shadow data set for a copy of the logical partition of the index, with a name that has the form `catname.DSNDBx.dbname.psname.S0mmm.Annn`.

If a table space, partition, or index resides in DB2-managed data sets and does not already exist when REORG is invoked, DB2 creates the shadow data sets. At the end of REORG processing, the DB2–managed shadow data sets are deleted.

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 invoke REORG with SHRLEVEL REFERENCE or SHRLEVEL CHANGE.

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. 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 non-committing 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.

Recovery Pending Status: You cannot reorganize a table space if:

- Any partition of the partitioned table space is in the recovery pending status
- The clustered index is in the recovery 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 recovery pending status
- The corresponding partitioned index is in the recovery pending status, and the data is unloaded by the cluster index method.

You cannot reorganize an index if any partition of the index is in the recovery pending status. Similarly, you cannot reorganize a single index partition if it is in the recovery pending status.

There are three types of recovery pending restrictive states:

RECP The table space, index space, or partition is in a recovery pending state. 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.

RECP* The logical partition of the type 2 nonpartitioned index is in a recovery pending status. The entire nonpartitioned index is inaccessible. RECP* can be reset by recovering only the single logical partition.

PSRCP The page set is in a recovery pending state. To recover, the entire page set must be recovered. Recover of a logical partition is prohibited.

See “Resetting Check Pending Status” on page 2-42 for information about resetting the recovery pending status.

Check Pending Status: You cannot reorganize a table space that is in the check pending status. Similarly, you cannot reorganize an index when the data is in the check pending status. See “Chapter 2-4. CHECK DATA” on page 2-37 for more information about resetting the check pending status.

Instructions for Specific Tasks

For the tasks described, specify the options and values shown with your utility control statement. The following tasks are described here:

Product-sensitive Programming Interface

Determining When a Table Space Should Be Reorganized

| 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 need to be reorganized. 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.

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 number 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 could 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. It is a good practice to run RUNSTATS after using the REORG utility if you want current statistics.

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.

With type 2 indexes there is a possibility that reorganization is needed less often. With type 1 indexes, pages are sometimes split and half of the space is lost. You need to run the REORG utility to regain the space. Type 2 indexes generally do not do page splits for inserts to the end of an index, so the space needs less reorganization. (Type 2 indexes containing nulls are an exception.)

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 a table space, index, or a partition of a table space or 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.
- 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 2-201 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 2-229 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.

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 2-151 for actions to take when you drop a table.

Considering 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.

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 need 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 needs to be reorganized, then the corresponding directory table space should also be reorganized:

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.

Limitations for Reorganizing the Catalog and Directory:

- You cannot reorganize DSNDB01.SYSUTILX.
- 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.

Associated Directory Table Spaces: When certain catalog table spaces are reorganized, you should reorganize the associated directory table space as well.

Catalog Table Space	Associated Directory Table Space
DSNDB06.SYSDBASE	DSNDB01.DBD01
DSNDB06.SYSPLAN	DSNDB01.SCT02
DSNDB06.SYSPKAGE	DSNDB01.SPT01

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, the user 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 2-202. 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

REORG can be temporarily paused. 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 a return code of 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, for user defined table spaces you may re-define the table space attributes. 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 could 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 yourself 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 invoked in the same step. The result could be reduced sort efficiency or an abend due to an out of space condition.

Improving Performance

To improve REORG performance:

- Run REORG concurrently on separate partitions of a partitioned table space. When you run REORG on partitions of a partitioned table space, the sum of each job's processor usage is greater than for a single REORG 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.

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.

- 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. However, if a job using SORTKEYS abends in the reload, sort, or build phase, it can only be restarted at the beginning of the reload phase.
- Specify SORTDATA on your REORG statement unless your data set is very large, and you do not want to allocate the extra DASD needed by DFSORT. In this case, DB2 unloads the data by table space scan and invokes DFSORT to sort the data into clustering order. SORTDATA is useful if either:
  - Your table's CLUSTERRATIO is less than 95% or
  - FAROFFPOS/CARD for your table is greater than 5%

  In general, the lower the CLUSTERRATIO of your tables, the greater the performance improvement of REORG when specifying SORTDATA.
- Specify NOSYSREC on your REORG statement. See “Omitting the Output Data Set” on page 2-208 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

Reviewing REORG Output

The output from REORG TABLESPACE consists of a reorganized table space or partition; from REORG INDEX it consists of a reorganized index or index partition. Table 41 summarizes the effect of REORG on a table space partition and on the corresponding index partition.

Table 41. REORG Summary

Specification	Results
REORG TABLESPACE	All data + entire partitioned index + all nonpartitioned indexes
REORG TABLESPACE PART*n*	Data for part *n* + part *n* of the partitioned index + index entries for part *n* in all nonpartitioned indexes
REORG INDEX	Entire index (all parts if partitioned)
REORG INDEX PART *n*	Part *n* of partitioned index

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

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 2-6. COPY” on page 2-57 for information on making image copies.
- If you have used SHRLEVEL NONE LOG NO, REORG places the reorganized table space or partition in copy pending status. At this time, perform a full

image copy to reset the copy pending status and to ensure that a backup is available for recovery.

- 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, so that the DB2 catalog statistics take into account the newly reorganized data, and SQL paths can be selected with accurate information.
- 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.

Considerations for Running REORG

This section discusses additional points to keep in mind when running REORG.

Sorting Data in Clustering Order

When SORTDATA is specified:

- 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 SHRLEVEL NONE or REFERENCE was specified (REORG cannot operate if a table has a clustering index and SHRLEVEL CHANGE is specified). 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 the SORTDATA option is specified, 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 the SORTDATA option is specified); 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 TERM UTILITY. Then recover the table space, using RECOVER, and run the REORG job again.

Reorganizing Partitioned Table Spaces

If you reorganize a single partition, all indexes of the table space are affected.
Depending on how disorganized the nonpartitioned 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 2-205.

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 REORG is run, 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 upon the UNLOAD option you specified on the original job:

- If you specified UNLOAD PAUSE, REORG sets a return code of 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 recovery pending status.

Terminating or Restarting REORG

If REORG is terminated by the TERM UTILITY command during the UNLOAD phase, objects have not yet been changed and the job can be rerun.

If REORG is terminated by 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 recovery pending status. Once the table space is recovered, the REORG job can be rerun.
- For SHRLEVEL REFERENCE or CHANGE, the data records are reloaded into shadow objects, so the original objects have not been affected by REORG. The job can be rerun.

If REORG is terminated by 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 recovery pending status. You can run REORG with the SORTDATA option or you can run RECOVER INDEX to recover those indexes.
- For SHRLEVEL REFERENCE or CHANGE, the records are reloaded into shadow objects, so the original objects have not been affected by REORG. The job can be rerun.

If REORG is terminated by 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. The job can be rerun. If there is a problem in renaming to the original data sets, then the objects are left in recovery pending status. The table space can then be recovered using the image copy created by REORG. The indexes must then also be recovered. After recovery, the objects have completed reorganization.

If REORG is terminated by the TERM UTILITY command during the BUILD2 phase, the logical partition is left in recovery pending status. After you run RECOVER INDEX for the logical partition, all objects have completed reorganization.

If you are restarting 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

Table 42. REORG Phases and Pending Statuses

Phase	Effect on Pending Status
UNLOAD	No effect
RELOAD	SHRLEVEL NONE has these effects: • Places table space in recovery pending status at the beginning of the phase and resets the status at the end of the phase. • Places indexes in recovery 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 recovery 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.
LOG	No effect
SWITCH	No effect. Under certain conditions, if TERM UTILITY is issued, it must complete successfully or objects may be placed in recovery pending status.
BUILD2	If TERM UTILITY is issued, the logical partitions for nonpartitioned indexes are placed in logical recovery pending status.

Recovering a Failed REORG Job: If REORG SHRLEVEL NONE is terminated 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 43 on page 2-218 provides information about restarting REORG TABLESPACE, depending on the phase REORG was in when the job stopped. Table 44 on page 2-219 provides information about restarting REORG INDEX.

If REORG is restarted in the UTILINIT phase, it is re-executed from the beginning of the phase. If REORG abends or system failure occurs while it is in the UTILTERM phase, it must be restarted 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 in order to perform the specified type of restart in the specified phase.

Table 43. 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
LOG			None None		
SWITCH		CURRENT PHASE	CURRENT PHASE	originals and shadows originals and shadows	3
BUILD2		CURRENT PHASE	CURRENT PHASE	shadows for nonpartitioned indexes shadows for nonpartitioned indexes	3,4 4

Note:

1. For NONE, if NOSYSREC is specified, then RESTART is not possible, and you must execute the RECOVER TABLESPACE utility for the table space or partition. For REFERENCE, if both SORTDATA and NOSYSREC are specified, then RESTART(CURRENT) or RESTART(PHASE) restarts at the beginning of the UNLOAD phase.
2. For NONE and REFERENCE, if SORTKEYS is specified, then RESTART(CURRENT) or RESTART(PHASE) restart at the beginning of the RELOAD phase.
3. The utility can be restarted with RESTART(CURRENT) or RESTART(PHASE). However, because this phase does not take checkpoints, RESTART restarts from the beginning of the phase.
4. If the PART option is specified with REORG TABLESPACE, the utility cannot be restarted at the beginning of the BUILD or BUILD2 phase if any nonpartitioned index is in a page set recovery pending (PSRCP) state.

Table 44. 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	SYSREC	
SORT	CURRENT PHASE	CURRENT PHASE	None None	SYSUT1 SYSUT1	1
BUILD	CURRENT PHASE	CURRENT PHASE	None None	SORTOUT SORTOUT	1
LOG			None None		
SWITCH		CURRENT PHASE	CURRENT PHASE	originals and shadows originals and shadows	1

Notes:

1 The utility can be restarted with either RESTART(CURRENT) or RESTART(PHASE). However, because this phase does not take checkpoints, RESTART is always re-executed from the beginning of the phase.

For instructions on restarting a utility job, see "Chapter 2-1. Invoking DB2 Online Utilities" on page 2-7.

Restarting REORG After an Out Of Space Condition: See "Restarting After an Out of Space Condition" on page 2-29 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 nonpartitioned type 2 indexes, are treated as distinct target objects. Utilities operating on different partitions of the same table space or index space are compatible. However, if a type 1 nonpartitioned index exists on a partitioned table space, utilities operating on different partitions of a table space can be incompatible because of contention on the nonpartitioned index.

REORG TABLESPACE Compatibility

Table 45 on page 2-220, Table 46 on page 2-221, Table 47 on page 2-221, and Table 48 on page 2-222 show which claim classes REORG drains and any restrictive state the utility sets on the target object.

For nonpartitioned type 2 indexes, REORG PART:

- Drains only the logical partition (and the repeatable read class for the entire index)
- Does not set the page set recovery pending status (PSRCP)
- Does not respect PCTFREE or FREEPAGE attributes when inserting keys

Table 49 on page 2-222 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 50 on page 2-223 shows which DB2 operations can be affected when reorganizing catalog table spaces.

For SHRLEVEL NONE, Table 45 on page 2-220 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 45. REORG TABLESPACE SHRLEVEL NONE. Use of claims and drains; restrictive states set.

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 or partition of 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
Nonpartitioned type 1 index	DW/UTRO	DA/UTUT	DW/UTRO	DA/UTUT
Nonpartitioned type 2 index	DW/UTRO	DA/UTUT		DR
Logical partition of nonpartitioned type 2 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 46 on page 2-221 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 46. REORG TABLESPACE SHRLEVEL REFERENCE. Use of claims and drains; restrictive states set.

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
Nonpartitioned type 1 index	DW/UTRO	DA/UTUT	DW/UTRO	DA/UTUT	
Nonpartitioned type 2 index	DW/UTRO	DA/UTUT		DR	
Logical partition of nonpartitioned type 2 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 47 shows which claim classes REORG drains and any restrictive state the utility sets on the target object.

Table 47. REORG TABLESPACE SHRLEVEL CHANGE. Use of claims and drains; restrictive states set.

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

[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 48 on page 2-222 shows which claim classes REORG drains and any restrictive state the utility sets on the target object.

Table 48. REORG TABLESPACE SHRLEVEL CHANGE. Use of claims and drains; restrictive states set.

Target	UNLOAD phase	Last iteration of LOG phase	SWITCH phase	BUILD2 phase
Partition of table space	CR/UTRW	DW/UTRO	DA/UTUT	UTRW
Partition of partitioning index	CR/UTRW	DW/UTRO	DA/UTUT	UTRW
Nonpartitioned type 1 index	CR/UTRW		DA/UTUT	
Nonpartitioned type 2 index			DR	
Logical partition of nonpartitioned type 2 index	CR/UTRW	DW/UTRO	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

Table 49 (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
COPY	No	Yes	Yes
DIAGNOSE	Yes	Yes	Yes
LOAD	No	No	No
MERGECOPY	No	No	No
MODIFY RECOVERY	No	No	No
QUIESCE	No	Yes	Yes
RECOVER INDEX	No	Yes	No
RECOVER TABLESPACE	No	No	No
REORG INDEX	No	Yes	No

Table 49 (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
REORG SHRLEVEL NONE UNLOAD CONTINUE or PAUSE, REORG SHRLEVEL REFERENCE, or REORG SHRLEVEL CHANGE	No	No	No
REORG SHRLEVEL NONE UNLOAD ONLY	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 50. 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

REORG INDEX Compatibility

Table 51 on page 2-224 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 52 on page 2-225 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 51. REORG INDEX. Use of claims and drains; restrictive states set.

Phase	REORG INDEX SHRLEVEL NONE	REORG INDEX SHRLEVEL REFERENCE	REORG INDEX SHRLEVEL CHANGE
UNLOAD	DW/UTRO	DW/UTRO	CR/UTRW
SORT	DA/UTUT (Type 1)		
BUILD	DA/UTUT (Type 2)		
Last iteration of LOG	n/a	n/a	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.

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

Table 52. REORG INDEX Compatibility

Action	REORG INDEX SHRLEVEL NONE REFERENCE, or CHANGE
CHECK DATA	No
CHECK INDEX	No
COPY	Yes
DIAGNOSE	Yes
LOAD	No
MERGECOPY	Yes
MODIFY RECOVERY	Yes
QUIESCE	No
RECOVER INDEX	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 SHRLEVEL NONE UNLOAD CONTINUE or PAUSE, REORG SHRLEVEL REFERENCE, or REORG SHRLEVEL CHANGE	No
REORG SHRLEVEL NONE UNLOAD ONLY without cluster index	Yes
REORG SHRLEVEL NONE UNLOAD ONLY 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 TABLESPACE	Yes
RUNSTATS INDEX	No
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.

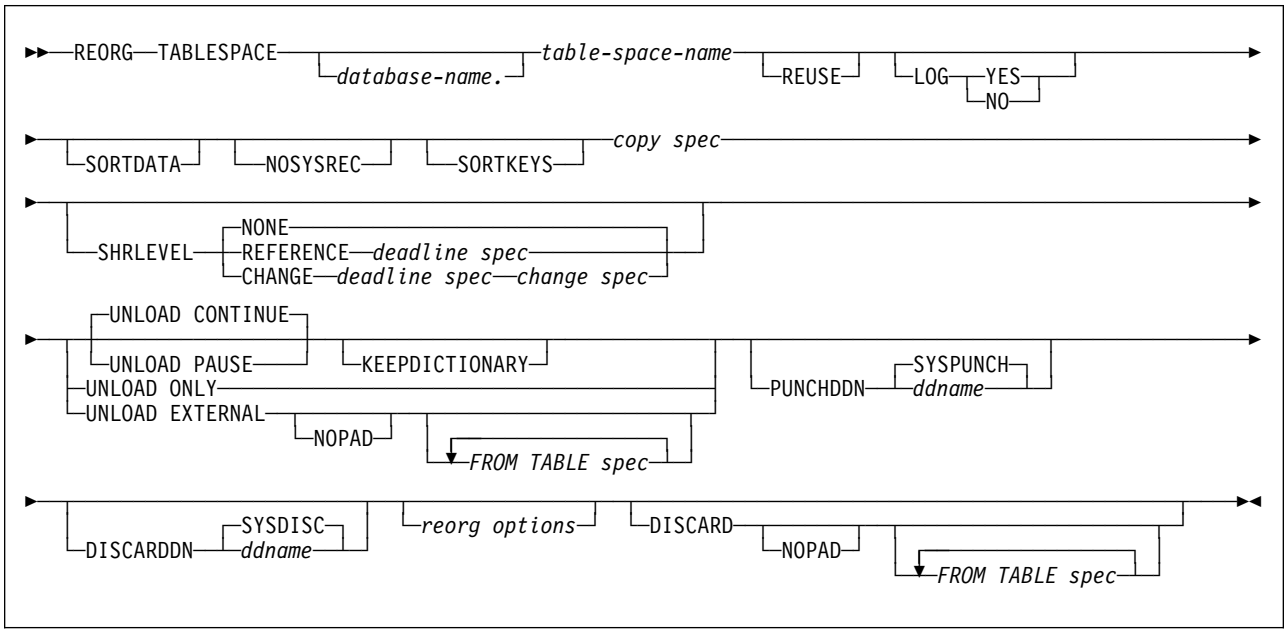
Syntax and Options of the Control Statement

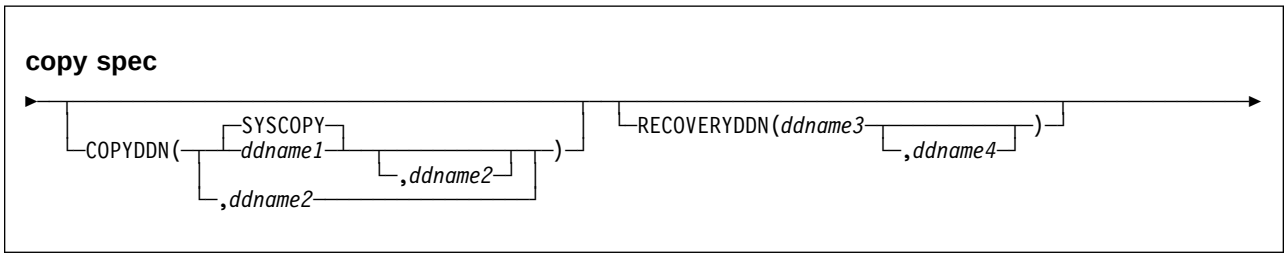
The utility control statement defines the function the utility job performs. You can create a control statement with the ISPF/PDF edit function. After creating it, save it in a sequential or partitioned data set. When you create the JCL for running the job, use the SYSIN DD statement to specify the name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

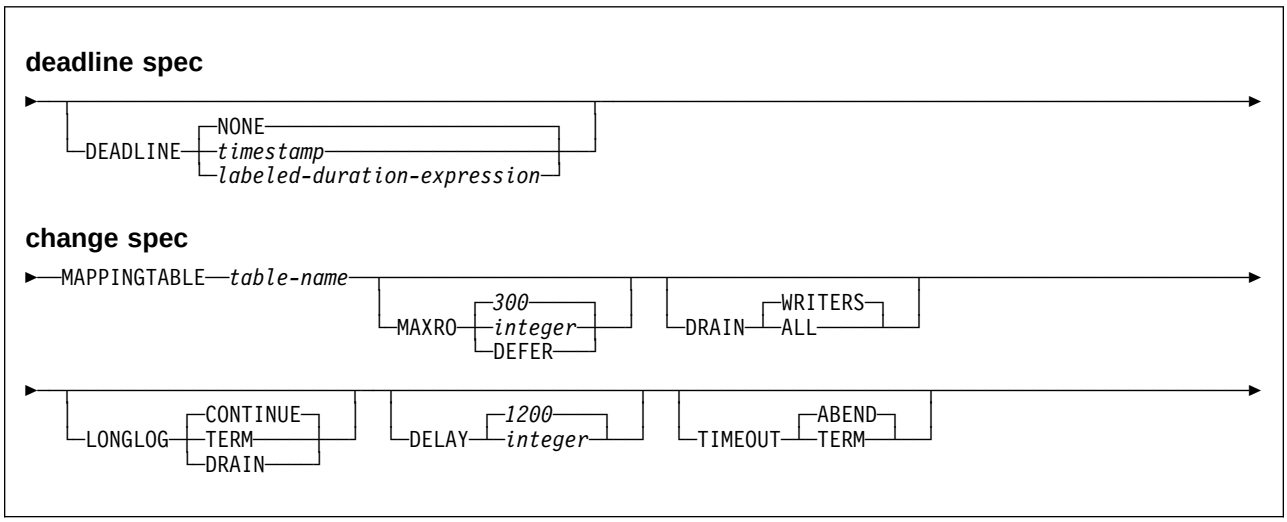
Syntax Diagram

For guidance in interpreting syntax diagrams, see “How to Read the Syntax Diagrams” on page 1-3.

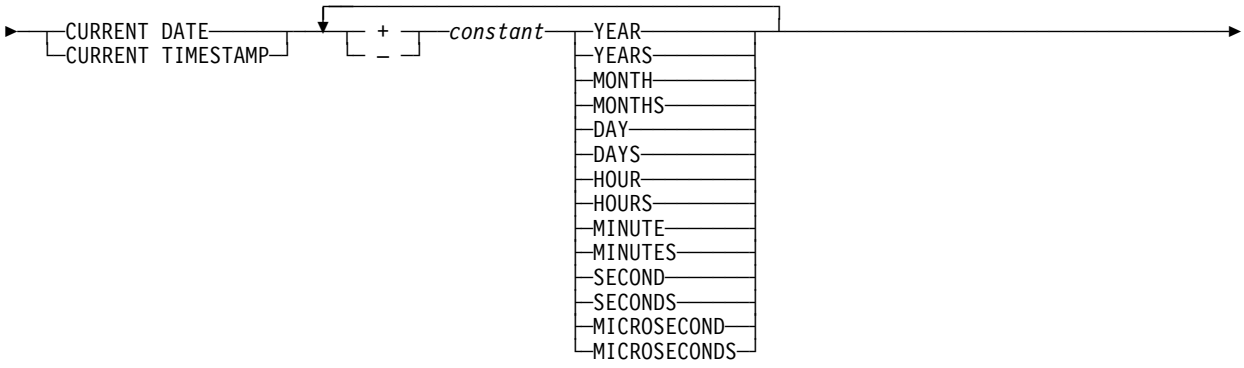
REORG TABLESPACE Syntax



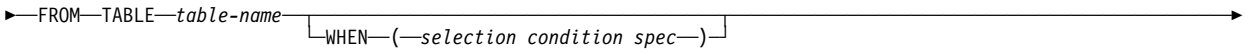




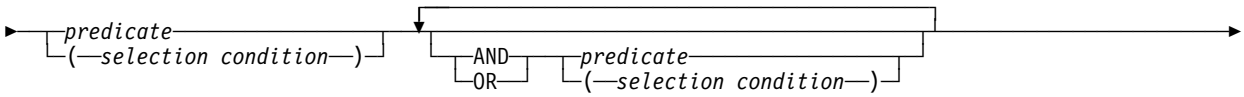
labeled-duration-expression:



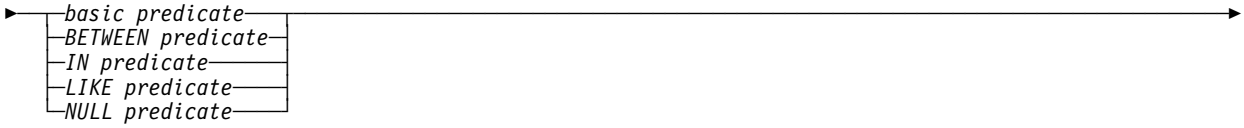
FROM TABLE spec:



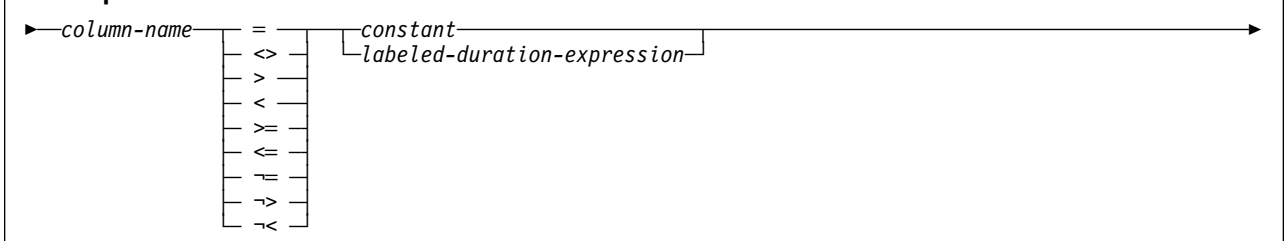
selection condition spec:



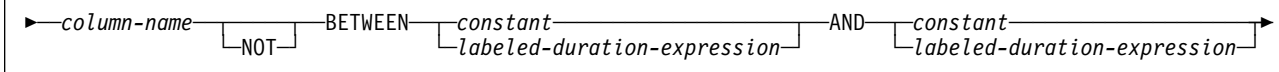
predicate:



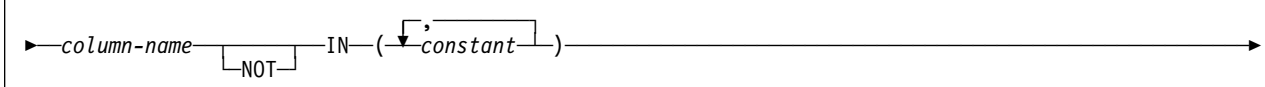
basic predicate:



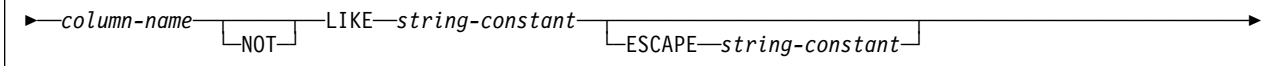
BETWEEN predicate:



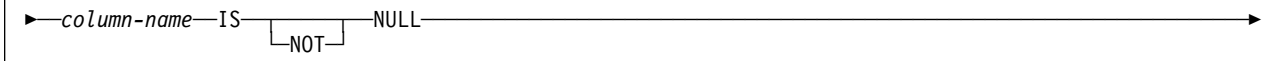
IN predicate:



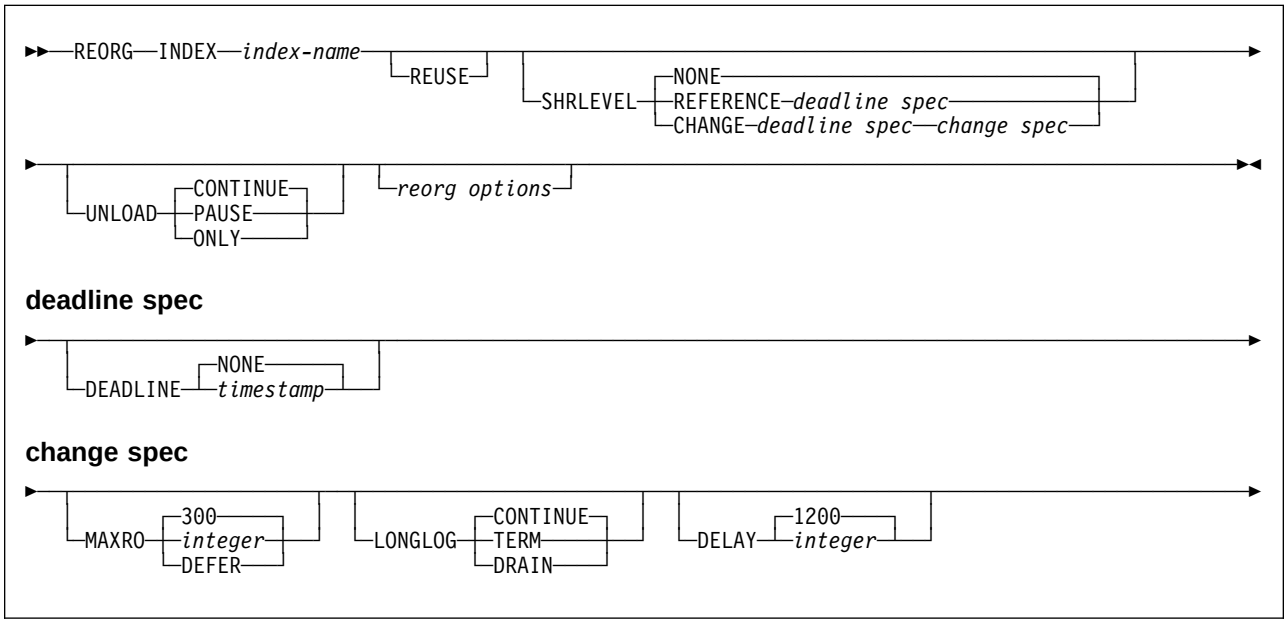
LIKE predicate:



NULL predicate:



REORG INDEX Syntax



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 2-7.

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.

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, DB2 deletes and redefines DB2-managed data sets to reset them.

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 SHRLEVEL NONE is specified or implied, the **default** is **LOG YES**.

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 2-209.

NOSYSREC
Specifies that the output of sorting (if there is a clustering index and SORTDATA is specified) 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 2-208 for addititional information about using this option.

SORTKEYS
Specifies that index keys will be sorted in parallel with the reload and build phases 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*) copied 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 state regardless of which LOG option is specified.

If SHRLEVEL NONE is specified (explicitly or by default) for REORG, and COPYDDN is not specified, then no image copy is created at the local site.

If SHRLEVEL REFERENCE or CHANGE is specified for REORG, and COPYDDN is not specified, then COPYDDN(SYSCOPY) is assumed, and a DD statement for SYSCOPY is required.

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.

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), reloading into that area (while applications have no access), and then allowing read/write access again.

The default is **NONE**.

If NONE is specified (explicitly or by default), the following parameters cannot be specified: MAPPINGTABLE (for REORG TABLESPACE), MAXRO, LONGLOG, DELAY, and DEADLINE. For REORG TABLESPACE, if SORTDATA is omitted, UNLOAD PAUSE is specified, or UNLOAD ONLY is specified, then NOSYSREC cannot be specified.

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 invoke REORG SHRLEVEL REFERENCE, see “Creating JCL Statements for Required Data Sets” on page 2-199.

If REFERENCE is specified, the following parameters cannot be specified:

- LOG (for REORG TABLESPACE). 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 (for REORG TABLESPACE), 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), reloading into a shadow copy of that area (while applications have read/write access to the orginal 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 invoke REORG SHRLEVEL CHANGE, see “Creating JCL Statements for Required Data Sets” on page 2-199.

If CHANGE is specified, the following parameters cannot be specified:

- LOG (for REORG TABLESPACE). Reorganization with CHANGE always creates an image copy and always refrains from logging records during reloading.
- SORTDATA, NOSYSREC, SORTKEYS (for REORG TABLESPACE). Reorganization with CHANGE always operates as if these parameters were specified.
- UNLOAD. Reorganization with CHANGE always performs UNLOAD CONTINUE.

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 2-201.

MAPPINGTABLE cannot be specified with REORG INDEX.

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 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 upon 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.

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 invoked.

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 + 3 DAYS + 6 HOURS + 30 MINUTES
```

UNLOAD
Specifies whether the utility job must continue processing or end after the data has been 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 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. 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 needs to be extracted 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. 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 UNLOAD ONLY is specified with REORG TABLESPACE, any edit routine or field procedure is invoked 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.

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.

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

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 following LOAD statement:

```
#                    LOAD DATA INDDN SYSREC   LOG NO  RESUME YES
#                     EBCDIC CCSID(00500,00000,00000)
#                     INTO TABLE "DSN8510 "."EMP               "
#                     WHEN(00004:00005 = X'0012')
#                     ( "EMPNO             " POSITION(00007:00012) CHAR(006)
#                     , "FIRSTNME          " POSITION(00013)       VARCHAR
#                     , "MIDINIT           " POSITION(*)           CHAR(001)
#                     , "LASTNAME          " POSITION(*)           VARCHAR
#                     ,  DSN_NULL_IND_00005  POSITION(*)           CHAR(1)
#                     , "WORKDEPT          " POSITION(*)           CHAR(003)
#                                              NULLIF(DSN_NULL_IND_00005)=X'FF'
#                     ,  DSN_NULL_IND_00006  POSITION(*)           CHAR(1)
#                     , "PHONENO           " POSITION(*)           CHAR(004)
#                                              NULLIF(DSN_NULL_IND_00006)=X'FF'
#                     ,  DSN_NULL_IND_00007  POSITION(*)           CHAR(1)
#                     , "HIREDATE          " POSITION(*)           DATE EXTERNAL
#                                              NULLIF(DSN_NULL_IND_00007)=X'FF'
#                     ,  DSN_NULL_IND_00008  POSITION(*)           CHAR(1)
#                     , "JOB               " POSITION(*)           CHAR(008)
#                                              NULLIF(DSN_NULL_IND_00008)=X'FF'
#                     ,  DSN_NULL_IND_00009  POSITION(*)           CHAR(1)
#                     , "EDLEVEL           " POSITION(*)           SMALLINT
#                                              NULLIF(DSN_NULL_IND_00009)=X'FF'
#                     ,  DSN_NULL_IND_00010  POSITION(*)           CHAR(1)
#                     , "SEX               " POSITION(*)           CHAR(001)
#                                              NULLIF(DSN_NULL_IND_00010)=X'FF'
#                     ,  DSN_NULL_IND_00011  POSITION(*)           CHAR(1)
#                     , "BIRTHDATE         " POSITION(*)           DATE EXTERNAL
#                                              NULLIF(DSN_NULL_IND_00011)=X'FF'
#                     ,  DSN_NULL_IND_00012  POSITION(*)           CHAR(1)
#                     , "SALARY            " POSITION(*)           DECIMAL
#                                              NULLIF(DSN_NULL_IND_00012)=X'FF'
#                     ,  DSN_NULL_IND_00013  POSITION(*)           CHAR(1)
#                     , "BONUS             " POSITION(*)           DECIMAL
#                                              NULLIF(DSN_NULL_IND_00013)=X'FF'
#                     ,  DSN_NULL_IND_00014  POSITION(*)           CHAR(1)
#                     , "COMM              " POSITION(*)           DECIMAL
#                                              NULLIF(DSN_NULL_IND_00014)=X'FF'
#                      )
```

PUNCHDDN *ddname*

Specifies the DD statement for a dataset 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 dataset.

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 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. A FROM TABLE statement is required to qualify the rows to be unloaded or discarded.

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 and OR) to the result of each specified predicate. If logical operators are not specified, the result of the selection condition is the result of the specified predicate.

Selection conditions within parentheses are evaluated first. If the order of evaluation is not specified by parentheses, AND is applied before OR.

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 2-243.

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

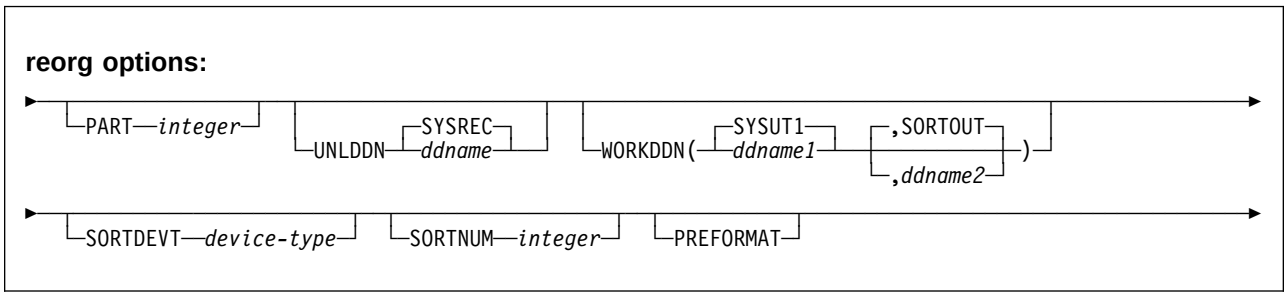
The *NULL predicate* tests for null values.

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 Options

For the descriptions of keywords and parameters included within *reorg options*, see page 2-244.

REORG Options Syntax



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 2-7.

PART *integer*

Identifies a partition to be reorganized. You can reorganize a single partition of a partitioned index or a partitioned table space.

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.

If PART is omitted, the entire table space or index is reorganized.

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***)**

For REORG TABLESPACE, *ddname* specifies the DD statement for a temporary data set used for intermediate output. For REORG INDEX, *ddname* specifies the DD statement for the unload data set.

ddname1

Is the DD name of the temporary work file for sort input. A work data set for sort input is required for tables with indexes unless SORTKEYS is specified.

The **default** is **SYSUT1**.

ddname2

Is the DD name of the temporary work file for sort output.

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 SORTKEYS is specified. If WORKDDN is not specified, or if it is specified 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 2-209.

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 a sort of the index keys is required, 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 2-209.

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 2-209.

PREFORMAT
Specifies that the remaining pages are preformatted up to the high allocated RBA in the table space and indexspaces 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 UNLOAD ONLY is specified.

For more information on PREFORMAT, see "Improving Performance with PREFORMAT" on page 2-102

Sample JCL and Control Statements

Examples

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.

```
//UTILSAMP JOB USER=SYSADM,PASSWORD=SYSADM,NOTIFY=SYSADM,
//         REGION=4M,TIME=1440,MSGCLASS=A,CLASS=A
//*
//*        UTILITY SAMPLE JOB
//*
//UTILSAMP EXEC PGM=DSNUTILB,REGION=1024K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*
//*********************************************************************
//*  DATA SETS USED BY THE UTILITY                                    *
//*********************************************************************
//SORTOUT  DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTWK01 DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//UNLD     DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//WORK     DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSPRINT DD  SYSOUT=*
//UTPRINT  DD  SYSOUT=*
//SYSIN    DD  *

 REORG TABLESPACE (DSN8D51A.DSN8S51D)
         UNLDDN (UNLD)
         WORKDDN (WORK)

//*
```

Example 2: Reorganizing a Table Space: Reorganize table space DSN8S51D in database DSN8D51A.

```
REORG TABLESPACE DSN8D51A.DSN8S51D
```

Example 3: Reorganizing a Table Space Partition: Reorganize partition 3 of table space DSN8S51E in database DSN8D51A.

```
REORG TABLESPACE DSN8D51A.DSN8S51E
  PART 3
  SORTDEVT SYSDA
```

Example 4: Reorganizing an Index: Reorganize index XMSGTXT1. Stop the utility after the index keys have been unloaded, but allow for subsequent restart.

```
REORG INDEX DSN8510.XMSGTXT1
  UNLOAD PAUSE
```

Example 5: REORG with DFSORT Unloading by Table Space Scan: Reorganize table space DSN8S51E in database DSN8D51A. Specify that DFSORT unloads the data by table space scan.

```
REORG TABLESPACE DSN8D51A.DSN8S51E SORTDATA
```

Example 6: REORG Using SORTKEYS: Use the SORTKEYS option to improve performance for a reorganization of the table space containing DSN8S51D.DEPT:

```
REORG TABLESPACE DSN8D51A.DSN8S51D SORTDATA SORTKEYS
```

Example 7: Reorganizing a Table while Allowing Read-Only Access: Reorganize table space DSN8S51D in database DSN8D51A. The deadline for start of the SWITCH phase is *3:15* on *February 4, 1997*.

```
REORG TABLESPACE DSN8D51A.DSN8S51D 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 DSN8S51D in database DSN8D51A. 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 DSN8D51A.DSN8S51D COPYDDN(MYCOPY1)
  RECOVERYDDN(MYCOPY2) SHRLEVEL CHANGE
  DEADLINE 1997-2-4-03.15.00
  MAPPINGTABLE MYMAPTABLE MAXRO 240 LONGLOG DRAIN DELAY 900
```

Chapter 2-15. 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.

Syntax Diagram: For a diagram of REPAIR syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 2-257.

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 use only to appropriate people.

Instructions for Running REPAIR

In order to run REPAIR, you must:

1. Read “Before Running REPAIR” on page 2-251 in this chapter.
2. Prepare the necessary data sets, as described in “Phases and Data Flow” on page 2-250.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for REPAIR, see “Sample JCL and Control Statements” on page 2-267.)
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 2-251. (For a complete description of the syntax and options for REPAIR, see “Syntax and Options of the Control Statement” on page 2-257.)
5. Check the compatibility table in “Concurrency and Compatibility” on page 2-254 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 REPAIR” on page 2-254.

Attention: Be extremely careful when using the REPAIR utility to replace data. Changing data to invalid values using REPLACE could produce unpredictable

results, particularly when changing page header information. Improper use of REPAIR can result in damaged data, or in some cases, system failure.

Invoking REPAIR

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for an explanation of ways to invoke DB2 utilities.

Phases and Data Flow

One of the following phases can be identified if the job terminates.

The phases for REPAIR are:

Phase	Description
UTILINIT	Initialization
REPAIR	
UTILTERM	Cleanup.

Creating JCL Statements for Required Data Sets

Table 53 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 53. 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 objects are named in the utility control statement and do not require DD cards 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 2-257 for REPAIR syntax and option descriptions. See “Sample JCL and Control Statements” on page 2-267 for examples of REPAIR usage.

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

Unless a table space is very large, use RECOVER INDEX to restore damaged index data. 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.

Instructions for Specific Tasks

For the tasks described, specify the options and values shown with your utility control statement. The following tasks are described here:

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 recovery 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 constraint by running CHECK DATA. CHECK DATA performs a complete check of all referential integrity constraints of the table space set, REPAIR leaves you with the responsibility of checking all the referential integrity constraints violations.

Repairing a Damaged Page

1. Invoke 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 Service, if available, for assistance in analyzing this information.
5. You could 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 workfile database, run REPAIR DBD DROP or DROP DATABASE (SQL) and then recreate it. If you receive errors when you drop the workfile database, contact your IBM Support Center for assistance.

Locating Rows by Key

If LOCATE TABLESPACE KEY is used, 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.

Writing Page Numbers for Partitioned Table Spaces: If the table space is partitioned, the byte string in PAGE X'*byte-string*' designates the partition number in certain high-order bits and the page number in the low-order bits. The length of the byte string is 32 bits (4 bytes) for large tables or 24 bits (3 bytes) for tables that are not large. To code the partition and page number in the appropriate byte string, follow these steps:

1. Find the page size (either 4 KB or 32 KB), table space type (either not large or large), and the number of partitions for the table space. This information appears in catalog table SYSIBM.SYSTABLESPACE. The page size is in the PGSIZE column, the table type (**L** for large, blank for not large) is in the TYPE column, and the number of partitions is in the PARTITION column.
2. Using this information, find in Table 54 the number of high-order bits in the PAGE byte string used to specify the partition number. For example, a large table with 4 KB pages has the partition number in the high-order 12 bits of the 32–bit string.

Table 54. Number of High-Order Bits Used for Partition Number

Type of Table Space	Size of PAGE Byte String	Number of Partitions	Partition Bits for 4KB Page	Partition Bits for 32KB Page
Not Large	24 bits (3 bytes)	1 to 16	4	7
Not Large	24 bits (3 bytes)	17 to 32	5	8
Not Large	24 bits (3 bytes)	33 to 64	6	9
Large	32 bits (4 bytes)	1 to 254	12	15

3. Code the partition number in the appropriate number of high-order bits. Fill the remainder with zeros.

   Attention: The first partition is numbered zero in DB2, so a partition number can range from 00 to 253.

   For example, to code the 22nd partition of a table that is not large and that has a page size of 4 KB, code the number 21 in the 5 high-order bits of a 24–bit string.

   ```
   10101000 00000000 00000000
   ```

   To code the 175th partition of a large table with a page size of 32 KB, code the number 174 in the 15 high-order bits of a 32–bit string.

   ```
   00000001 01011100 00000000 00000000
   ```

4. Code the page number in the low-order 24 bits of the appropriate bit string (24 or 32 bits). That is, code it as a 24–bit binary number.
5. Add the two strings and convert the result to hexadecimal notation. Use the hexadecimal value as the value for the PAGE keyword. For example, if the page number in the 175th partition for the large table case was page number 255, the resulting binary string would be:

   ```
   00000000 01011100 00000000 11111111
   ```

Convert that binary number to “01FA00FF” in hexadecimal. Write the PAGE option as PAGE X'01FA00FF'.

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.

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: While 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 2-37 for information about resetting this status.

Terminating or Restarting REPAIR

REPAIR can be terminated with the TERM UTILITY command. See Chapter 2 of *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. However, if a type 1 nonpartitioned index exists on a partitioned table space, utilities operating on different partitions of a table space can be incompatible because of contention on the nonpartitioned index.

Table 55 on page 2-255 shows which claim classes REPAIR drains and any restrictive state the utility sets on the target object.

Table 56 on page 2-255 and Table 57 on page 2-256 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 55. REPAIR. Use of claims and drains; restrictive states set.

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 56 (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
COPY	Yes	No
DIAGNOSE	Yes	Yes
LOAD	No	No
MERGECOPY	Yes	Yes
MODIFY	Yes	Yes
QUIESCE	Yes	No
RECOVER INDEX	No	No

Note: RECOVER INDEX is compatible with LOCATE by RID, DUMP or VERIFY.

Table 56 (Page 2 of 2). Compatibility with REPAIR, LOCATE by KEY or RID

	DUMP or VERIFY	DELETE or REPLACE
RECOVER TABLESPACE	No	No
REORG INDEX	No	No
Note: REORG INDEX is compatible with LOCATE by RID, DUMP, VERIFY, or REPLACE.		
REORG UNLOAD CONTINUE or PAUSE	No	No
REORG UNLOAD ONLY	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 TABLESPACE	Yes	No
RUNSTATS INDEX SHRLEVEL REFERENCE	Yes	No
RUNSTATS INDEX SHRLEVEL CHANGE	Yes	Yes
STOSPACE	Yes	Yes

Table 57 (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
COPY	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
RECOVER INDEX	Yes	No	No	No
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

Table 57 (Page 2 of 2). Compatibility with REPAIR, LOCATE by PAGE

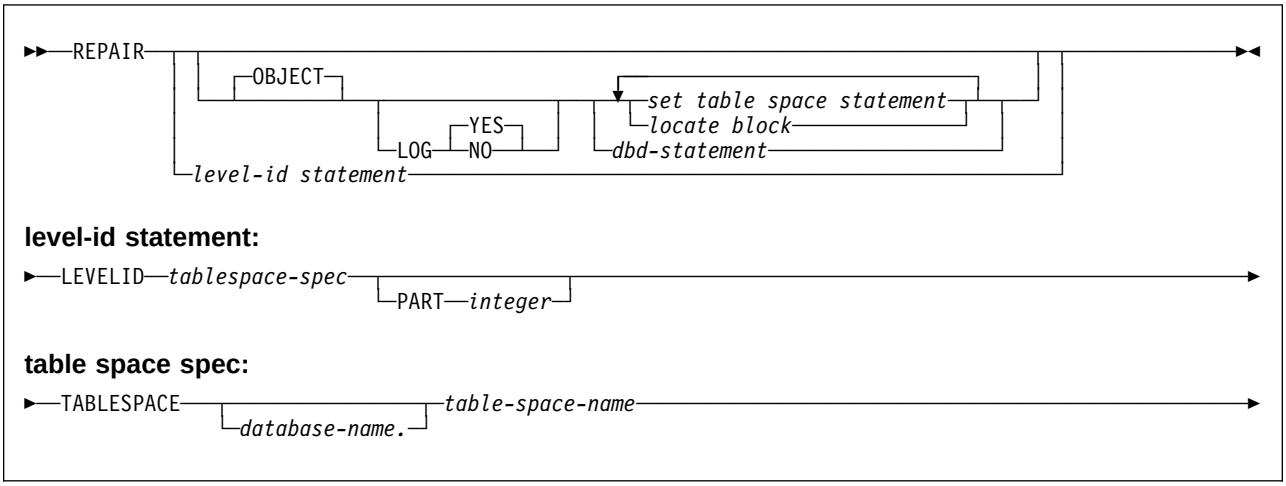
	TABLESPACE DUMP or VERIFY	TABLESPACE REPLACE	INDEX DUMP or VERIFY	INDEX REPLACE
REORG UNLOAD CONTINUE or PAUSE	No	No	No	No
REORG UNLOAD ONLY	Yes	No	Yes	Yes
REPAIR DUMP or VERIFY	Yes	No	Yes	No
REPAIR DELETE or REPLACE	No	No	No	No
Note: REPAIR LOCATE INDEX PAGE REPLACE is compatible with LOCATE TABLESPACE PAGE.				
REPORT	Yes	Yes	Yes	Yes
RUNSTATS TABLESPACE	Yes	No	Yes	Yes
RUNSTATS INDEX	Yes	Yes	Yes	No
STOSPACE	Yes	Yes	Yes	Yes

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 1-3.



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 2-7.

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**.

NO
Does not log the changes. You cannot use this option with a DELETE statement.

LEVELID
Sets the level identifier of the named table space or 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 or partition with outstanding indoubt log records or pages in the logical page list (LPL).

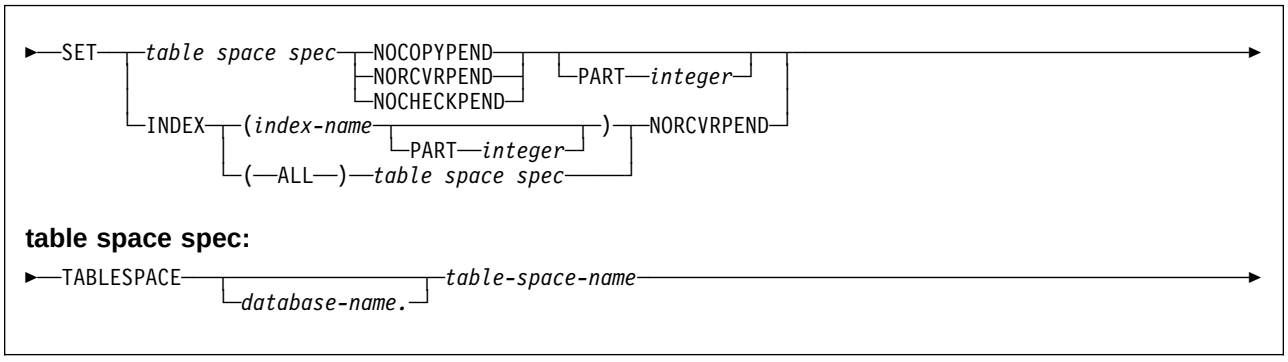
Attention: Accepting a down-level data set might cause data inconsistencies. Problems with inconsistent data resulting from resetting the level identifier are the responsibility of the user.

PART
Identifies a partition of the named table space.

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.

SET TABLESPACE and SET INDEX Statement Syntax

The SET TABLESPACE statement resets the copy pending, recovery pending, and check pending statuses for a table space or data set. The SET INDEX statement resets the “recovery 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
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.

INDEX
Specifies the index whose recovery pending status is to be reset.

(*index-name***)**
Names the index to be processed (set NORCVRPEND only).

(ALL)
Specifies that all indexes in the table space will be processed (set NORCVRPEND only).

The keyword INDEXES is still accepted following a *table space spec* and causes all indexes to be processed (set NORCVRPEND only). All indexes can also be processed (set NORCVRPEND only) by specifying INDEX(ALL) followed by a *table space spec*.

NOCOPYPEND
Resets the copy pending status of the specified table space.

NORCVRPEND
Resets the recovery pending status of the specified table space or index.

NOCHECKPEND
Resets the check pending status of the specified table space.

PART *integer*
Specifies a particular partition whose copy pending or recovery pending status is to be reset. If you do not specify PART, REPAIR resets the pending state of the entire table space or index.

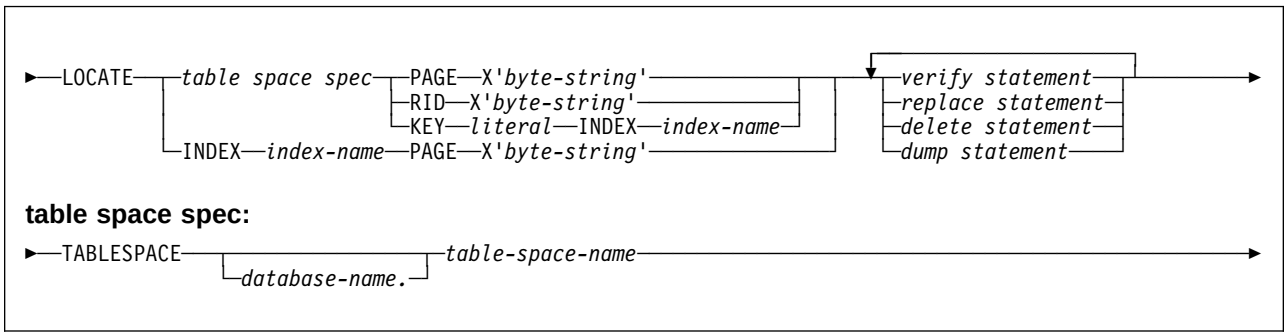
integer is the number of the partition and must be in the range from 1 to the number of partitions defined for the table space. The maximum is 254.

PART can be specified for NOCHECKPEND on a table space, and for NORCVRPEND on indexes.

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.

PAGE X'*byte-string***'**
Specifies that the data of interest is an entire page. The offsets given in *byte-string* and in later statements are relative to the beginning of the page. The first byte of the page is 0.

byte-string can be 1 to 6 hexadecimal digits. You need not enter leading zeros. For the method of writing page numbers in partitioned table spaces, see page 2-253. Enclose the byte string between apostrophes and precede it with X.

RID X'*byte-string***'**
Specifies that the data of interest is a single row. The offsets given in *byte-string* and in later statements are relative to the beginning of the row. The first byte of the stored row prefix is 0.

byte-string can be 1 to 8 hexadecimal digits. You do not need to enter leading zeros. Enclose the byte string between apostrophes and precede it with an X.

KEY *literal*
Specifies that the data of interest is a single row, identified by *literal*. The offsets given in later 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 invoke 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.

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.

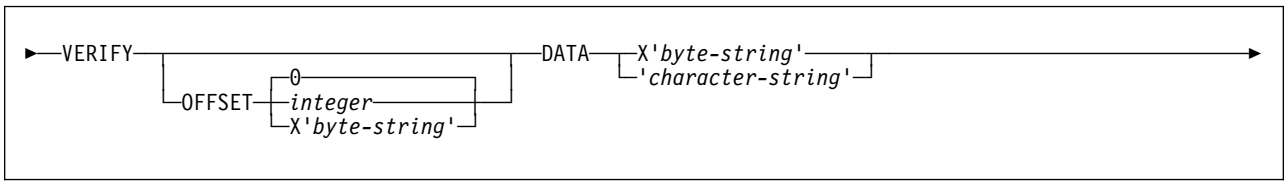
PAGE X'*byte-string***'**

Specifies that the data of interest is an entire page. The offsets given in *byte-string* and in later statements are relative to the beginning of the page. The first byte of the page is 0.

byte-string can be 1 to 6 hexadecimal digits. You need not enter leading zeros. For the method of writing page numbers in partitioned table spaces, see page 2-253. 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, later operations in the same LOCATE block are allowed to proceed. If *any* data area *does not* contain its specified value, *all* later 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 1 to 4 hexadecimal digits. You need not enter leading zeros. Enclose the byte string between apostrophes 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***'**

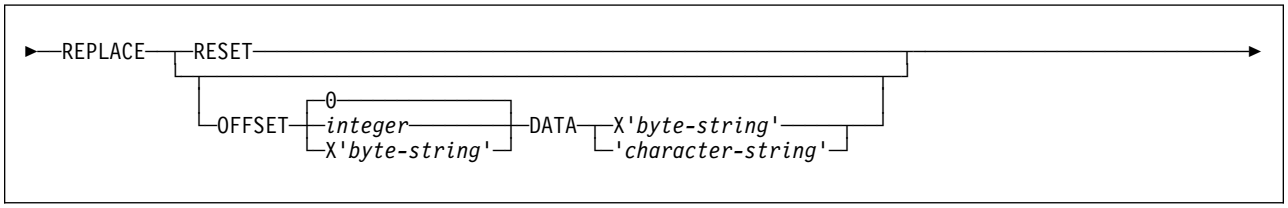
Can be an *even number*, from 2 to 32, of hexadecimal digits that must be present. You need not enter leading zeros. Enclose the byte string between apostrophes and precede it with X.

'*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 1 to 4 hexadecimal digits. You need not enter leading zeros. Enclose the byte string between apostrophes and precede it with X.

DATA

Defines the new data that is to be entered. Only one OFFSET and one DATA specification are acted upon 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 2 to 32, of hexadecimal digits to replace the current data. You need not enter leading zeros. Enclose the byte string between apostrophes and precede it with X.

'*character-string***'**

Can be any character string to replace the current data.

DELETE Statement Syntax and Description

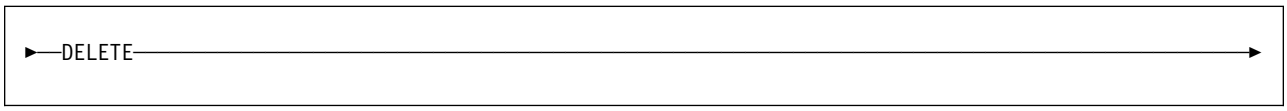
The DELETE statement deletes a single row of data that has been located by a RID or KEY option. The statement falls within a LOCATE block. If any VERIFY statement within that block finds a data area that does not contain its specified data, the DELETE operation is inhibited.

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 could lead to an unexpected result. See page 2-268 for a possible way to delete a child row without updating its predecessor and successor.

In any LOCATE block, you can use DELETE only once.

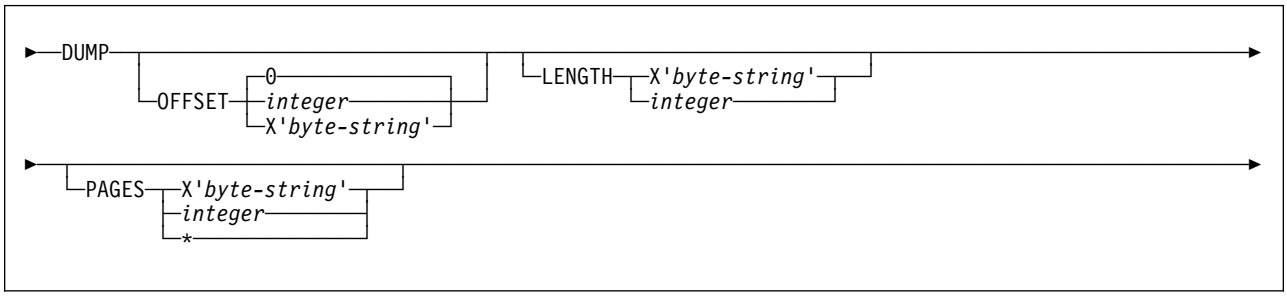
You cannot use DELETE if you have used any of these options:

- 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.



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.



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 1 to 4 hexadecimal digits. You need not enter leading zeros. Enclose the byte string between apostrophes 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 1 to 4 hexadecimal digits. You need not enter leading zeros. Enclose the byte string between apostrophes 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 1 to 4 hexadecimal digits. You need not enter leading zeros. Enclose the byte string between apostrophes 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.

DBD Statement Syntax

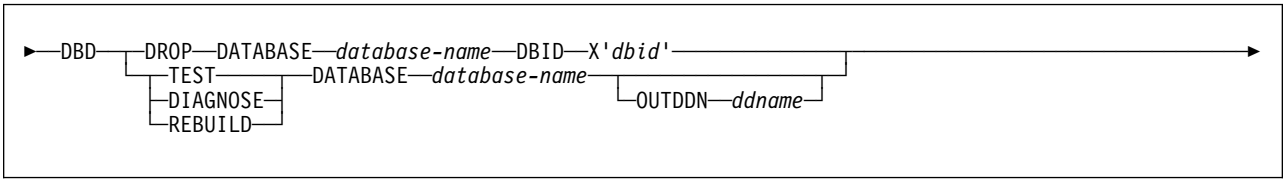
The DBD statement allows you to:

- 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
- Drop an inconsistent database definition from the DB2 catalog and the DB2 directory.
- Drops the specified index from the DB2 catalog and DB2 directory. Use this keyword if the SQL DROP INDEX statement fails because the description of the index is not in the catalog and directory. This keyword can also be used to drop an index when the index has the release dependency marker set to a higher release. See the *SQL Reference* for the affect of the DROP INDEX command.

  You can drop an index on a primary key but not a non-primary key that enforces RI with REPAIR DBD DROP INDEX.

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 3-37) 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.



DBD Statement Option Descriptions

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 Service, 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.

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.

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.

Sample JCL and Control Statements

Examples

Example 1: Replacing Damaged Data and Verifying Replacement: Repair the specified page of table space DSN8S51E. 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.

```
//UTILSAMP EXEC PGM=DSNUTILB,REGION=1024K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*
//*********************************************************************
//*  DATA SETS USED BY THE UTILITY                                    *
//*********************************************************************
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
  REPAIR OBJECT
    LOCATE TABLESPACE DSN8D51A.DSN8S51D 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'
```

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 2-16. 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. Use REPORT RECOVERY to find information necessary for recovering a table space.

Syntax Diagram: For a diagram of REPORT syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 2-274.

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 upon those tables.

The output from REPORT RECOVERY consists of the recovery history from the SYSIBM.SYSCOPY catalog table, log ranges from the SYSIBM.SYSLGRNX catalog table, and volume serial numbers where archive log data sets from the BSDS reside.

In a data sharing environment, the output provides:

- The RBA when migrated to Version 5
- 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.

Instructions for Running REPORT

In order to run REPORT, you must:

1. Prepare the necessary data sets, as described in “Creating JCL Statements for Required Data Sets” on page 2-270.
2. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for REPORT, see “Sample JCL and Control Statements” on page 2-276.)
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 2-270. (For a complete description of the syntax and options for REPORT, see “Syntax and Options of the Control Statement” on page 2-274.)

4. Check the compatibility table in “Concurrency and Compatibility” on page 2-274 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 2-274.

Invoking REPORT

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for an explanation of ways to invoke DB2 utilities.

Phases and Data Flow

The REPORT utility operates in these phases:

Phase	Description
UTILINIT	Initialization and setup
REPORT	Information collection
UTILTERM	Cleanup

Creating JCL Statements for Required Data Sets

Table 58 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 58. 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 objects are named in the utility control statement and do not require DD cards 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 2-274 for REPORT syntax and option descriptions. See “Sample JCL and Control Statements” on page 2-276 for examples of REPORT usage.

Instructions for Specific Tasks

For the tasks described, specify the options and values shown with your utility control statement. The following tasks are described here:

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 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 DSN8S51E:

```
REPORT TABLESPACESET TABLESPACE DSN8D51A.DSN8S51E
```

The following statement reports recovery information for table space DSN8S51D for the local subsystem only:

```
REPORT RECOVERY TABLESPACE DSN8D51A.DSN8S51D 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.

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.SYSUTIL
- 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.

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 upon those tables.

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 20. 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 needed 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 *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 21 on page 2-273 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 = 1995-04-06-09.38.21.734394,  IC TYPE = *Y*,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =AAE19AEEE0D4
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
DSNAME    = LDB1.TS1                                       ,   MEMBER NAME = V51A

TIMESTAMP = 1995-04-06-09.38.48.913881,  IC TYPE =  F ,  SHR LVL = R,  DSNUM    = 0000,  START LRSN =AAE19B0B107B
DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
DSNAME    = CPYLPF1                                        ,   MEMBER NAME = V51A

TIMESTAMP = 1995-04-06-09.39.09.542154,  IC TYPE = *Q*,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =AAE19B1EBCAF
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = W,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
DSNAME    = LDB1.TS1                                       ,   MEMBER NAME = V51A

TIMESTAMP = 1995-04-06-09.39.15.898561,  IC TYPE =  I ,  SHR LVL = R,  DSNUM    = 0000,  START LRSN =AAE19B24CC8A
DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
DSNAME    = CPYLPF2                                        ,   MEMBER NAME = V51A

TIMESTAMP = 1995-04-06-09.39.40.850896,  IC TYPE = *Q*,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =AAE19B3C986B
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = W,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
DSNAME    = LDB1.TS1                                       ,   MEMBER NAME = V51A

TIMESTAMP = 1995-04-06-09.39.47.840743,  IC TYPE =  I ,  SHR LVL = R,  DSNUM    = 0000,  START LRSN =AAE19B4342EC
DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
DSNAME    = CPYLPF3                                        ,   MEMBER NAME = V51A
DSNU586I = DSNUPSUM - REPORT RECOVERY TABLESPACE LDB1.TS1 SUMMARY
DSNU588I = DSNUPSUM - NO DATA TO BE REPORTED

DSNU592I = DSNUPREC - REPORT RECOVERY INFORMATION FOR DATA SHARING MEMBER : V51A
DSNU583I = DSNUPPLR - SYSLGRNX ROWS FROM REPORT RECOVERY FOR TABLESPACE LDB1.TS1
 UCDATE     UCTIME      START RBA      STOP  RBA     START LRSN     STOP  LRSN    PARTITION   MEMBER ID
 040695    16375569   000002017A57   000002019033   AAE19AD84EB1   AAE19AD8BA67     0000         0001
 040695    16380534   00000201C61C   000002022F16   AAE19AE1822F   AAE19AEE6DE6     0000         0001
 040695    16382567   AAE19AF4653C   AAE19AF4653C   AAE19AF4653C   AAE19AF4653C     0000         0001
 040695    16390230   0000020297AB   00000202FA14   AAE19B17C9D3   AAE19B1E7E32     0000         0001
 040695    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 : V51B


DSNU586I = DSNUPSUM - REPORT RECOVERY TABLESPACE LDB1.TS1 SUMMARY
DSNU588I = DSNUPSUM - NO DATA TO BE REPORTED

DSNU592I = DSNUPREC - REPORT RECOVERY INFORMATION FOR DATA SHARING MEMBER : V51C


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 21. Example of REPORT RECOVERY

Terminating or Restarting REPORT

You can terminate REPORT with the TERM UTILITY command.

For guidance in restarting online utilities, see “Restarting an Online Utility” on page 2-28.

Concurrency and Compatibility

REPORT does not set a utility restrictive state on the target table space or partition.

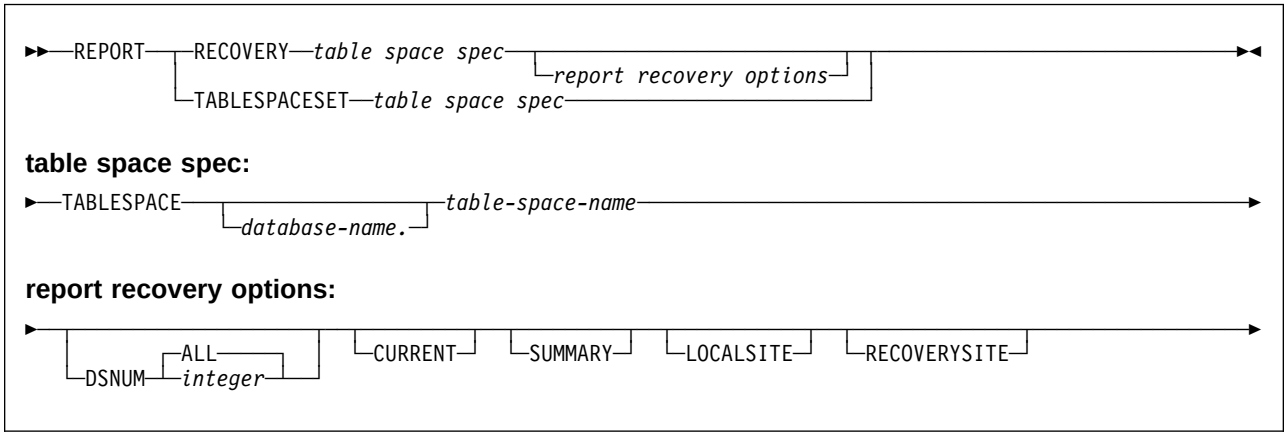
REPORT can run concurrently on the same target object with any utility or SQL operation.

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 1-3.



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 2-7.

RECOVERY
Indicates that recovery information for the specified table space is to be reported.

TABLESPACESET
Indicates that the names of all table spaces 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

Specifies the database the table space belongs to, and is optional.

table-space-name

Specifies the table space.

The following are optional:

DSNUM

Identifies a partition or data set, within the table space, for which information is to be reported; or it reports information for the entire table space.

ALL

Reports information for the entire table 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, 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.DSNDBx.dbname.tsname.I0001.An

where:

catname	The VSAM catalog name or alias
x	C or D
dbname	The database name
tsname	The table space name
n	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 DSNUM ALL is specified, then the last recoverable point is a full image copy that was taken for the entire table space.

If you do not specify CURRENT or if no last recoverable point exists, all SYSCOPY and SYSLGRNX entries for that table space are reported, including those on archive logs. However, if the CURRENT option is specified, 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.

Sample JCL and Control Statements

Examples

Example 1: Sample JCL for REPORT RECOVERY:

```
//UTILSAMP EXEC PGM=DSNUTILB,REGION=1024K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*
//*********************************************************************
//*  DATA SETS USED BY THE UTILITY                                    *
//*********************************************************************
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *

 REPORT RECOVERY
        TABLESPACE DSN8D51A.DSN8S51E
//*
```

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 DSN8D51A.DSN8S51E.

```
REPORT TABLESPACESET TABLESPACE DSN8D51A.DSN8S51E
```

Example 4: REPORT Recovery Information for a Table Space: This statement reports recovery information for table space DSN8D51A.DSN8S51D.

```
REPORT RECOVERY TABLESPACE DSN8D51A.DSN8S51D DSNUM ALL
```

Chapter 2-17. RUNSTATS

The RUNSTATS online utility gathers summary information about the characteristics of the data in table spaces, indexes, and partitions. This information is recorded in the DB2 catalog and is used by DB2 to select access paths to data during the bind process. It is available to the database administrator for evaluating database design and 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, indexes or columns; RUNSTATS INDEX gathers statistics only on indexes.

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.

Syntax Diagram: For a diagram of RUNSTATS syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 2-288.

Output: RUNSTATS updates the DB2 catalog with table space or index space statistics or prints a report. DB2 uses the information updated by RUNSTATS to select access paths to the data. You can query the catalog tables to obtain the updated statistics. See “Reviewing RUNSTATS Output” on page 2-280 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.

Instructions for Running RUNSTATS

In order to run RUNSTATS, you must:

1. Read “Before Running RUNSTATS” on page 2-278 in this chapter.
2. Prepare the necessary data sets, as described in “Creating JCL Statements for Required Data Sets” on page 2-278.
3. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for RUNSTATS, see “Sample JCL and Control Statements” on page 2-294.)

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 2-279. (For a complete description of the syntax and options for RUNSTATS, see “Syntax and Options of the Control Statement” on page 2-288.)
5. Check the compatibility table in “Concurrency and Compatibility” on page 2-286 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” on page 2-286. RUNSTATS can be restarted, but it starts over again from the beginning.

Invoking RUNSTATS

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for a description of ways to invoke DB2 utilities.

Phases and Data Flow

The RUNSTATS utility operates in these phases:

Phase	Description
UTILINIT	Initialization and setup
RUNSTATS	Scanning table space or index and updating catalog
UTILTERM	Cleanup

Creating JCL Statements for Required Data Sets

Table 59 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 59. 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 Diagram” on page 2-288 for RUNSTATS syntax and option descriptions. See “Sample JCL and Control Statements” on page 2-294 for examples of RUNSTATS usage

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. Since RUNSTATS puts information in these columns, values changed by the user are replaced.

Instructions for Specific Tasks

For the tasks described, specify the options and values shown with your utility control statement. The following tasks are described here:

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
- After there have been extensive updates, deletions, or insertions in a table space
- After you have run RECOVER TABLESPACE, RECOVER INDEX, or REORG INDEX

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

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.

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.

- COLCARD in SYSCOLUMNS
- 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 upon the source of the statistics as well as the value of the UPDATE option.

Table 60 (Page 1 of 2). Catalog Tables Updated by RUNSTATS

Keyword	UPDATE Option	Catalog Table
TABLESPACE	UPDATE ALL	SYSTABLESPACE SYSTABLEPART SYSTABLES SYSTABSTATS[1]
	UPDATE ACCESSPATH	SYSTABLESPACE SYSTABLES SYSTABSTATS[1]
	UPDATE SPACE	SYSTABLEPART
TABLE	UPDATE ALL	SYSCOLUMNS SYSCOLSTATS[1]
	UPDATE ACCESSPATH	SYSCOLUMNS SYSCOLSTATS[1]

Table 60 (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

Note: [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

Access Path Statistics

The catalog table columns listed in Table 61 are used by DB2 to select access paths to data during the bind process. Refer to Section 5 (Volume 2) of *Administration Guide* for further information regarding these columns.

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 “Notices” on page ix.

Table 61 (Page 1 of 2). 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
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 invoked to update the statistics.	G
SYSTABSTATS		
CARD	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.	S

Table 61 (Page 2 of 2). DB2 Catalog Columns Used to Select Data Access Paths

Column Name	Column Description	Use
HIGH2KEY	First 8 bytes of the second highest value of the column.	S
LOW2KEY	First 8 bytes of the second lowest value of the column.	S
STATSTIME	The date and time when RUNSTATS was last invoked 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 invoked to update the statistics.	G
SYSTABLESPACE		
NACTIVE	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.	S
STATSTIME	The date and time when RUNSTATS was last invoked to update the statistics.	G
SYSINDEXES		
CLUSTERRATIO	Percentage of rows in clustering order. For the clustering index, this ratio is optimal after reorganization. A row is considered in clustering order if the access of data is in ascending RID sequence (applicable to the first distinct key value). For nonclustering indexes, this value is determined by the correlation between this index and the clustering index (which can vary widely) after reorganization.	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 invoked to update the statistics.	G

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 “Notices” on page ix.

Table 62 (Page 1 of 4). DB2 Catalog Columns for Tuning Information

Column Name	Column Description	Use
SYSTABLEPART		
CARD	Total number of rows in the table space or partition. 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
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 needs 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

Table 62 (Page 2 of 4). DB2 Catalog Columns for Tuning Information

Column Name	Column Description	Use
PERCACTIVE	Percentage of space occupied by active rows, containing actual data from active tables. Other space is free, or occupied by dropped records. 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
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
SPACE	The number of kilobytes of space currently allocated for all extents.	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.	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 62 (Page 3 of 4). 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 could be attributed to the number of space map pages contained in the segmented table space.	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.	S
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

Table 62 (Page 4 of 4). DB2 Catalog Columns for Tuning Information

Column Name	Column Description	Use
SPACE	The number of kilobytes of space currently allocated for all extents.	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.	G

After Running RUNSTATS

After running RUNSTATS, rebind any application plans that use the tables or indexes so that they use the new statistics.

Terminating or Restarting

You can restart RUNSTATS with either a current or phase restart; however, RUNSTATS restarts from the beginning of the phase.

For more guidance in restarting online utilities, see “Restarting an Online Utility” on page 2-28.

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 type 1 nonpartitioned index exists on a partitioned table space, utilities operating on different partitions of a table space can be incompatible because of contention on the nonpartitioned index.

Table 63 on page 2-287 shows which claim classes RUNSTATS claims and drains and any restrictive state the utility sets on the target object.

Table 64 on page 2-287 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 63. RUNSTATS. Use of claims and drains; restrictive states set.

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*		
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.

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 64 (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
COPY	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
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 UNLOAD CONTINUE or PAUSE	No	No	No	No
REORG UNLOAD ONLY	Yes	Yes	Yes	Yes
REPAIR DUMP or VERIFY	Yes	Yes	Yes	Yes
REPAIR LOCATE KEY or RID DELETE or REPLACE	No	No	No	Yes

Table 64 (Page 2 of 2). RUNSTATS Compatibility

	RUNSTATS TABLESPACE SHRLEVEL REFERENCE	RUNSTATS TABLESPACE SHRLEVEL CHANGE	RUNSTATS INDEX SHRLEVEL REFERENCE	RUNSTATS INDEX SHRLEVEL CHANGE
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

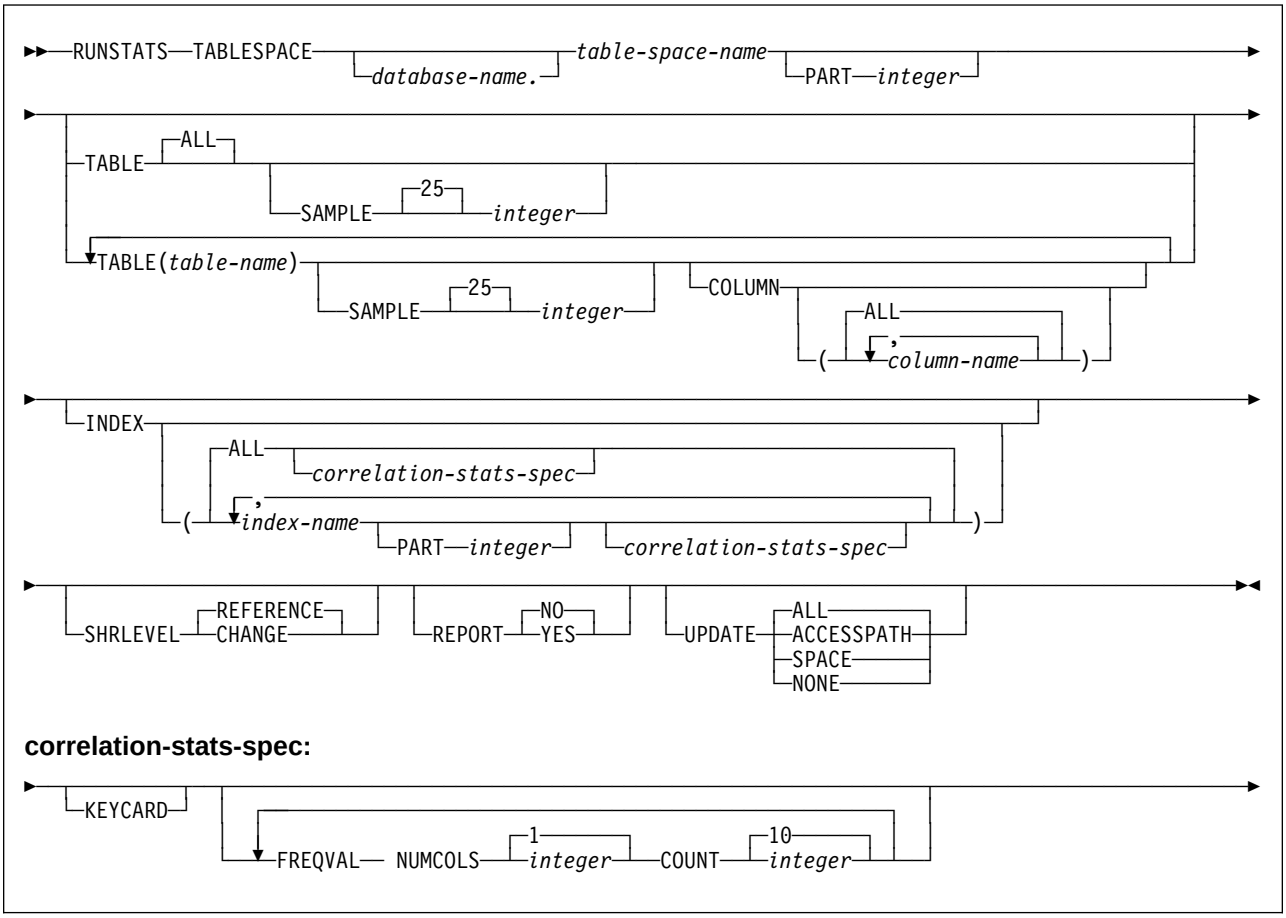
Syntax and Options of the Control Statement

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 1-3.

RUNSTATS TABLESPACE Syntax



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 2-7.

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.

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**.

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.

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.

KEYCARD
Collect key cardinalities on the concatenation of key columns for the specified indexes.

FREQVAL
Controls the collection of frequent value statistics. If FREQVAL is specified, then it must be followed by two additional options:

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*.

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 upon the combination of keywords (such as TABLESPACE, INDEX, TABLE, and COLUMN) specified with the RUNSTATS utility. However, these messages are *not* dependent upon 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.

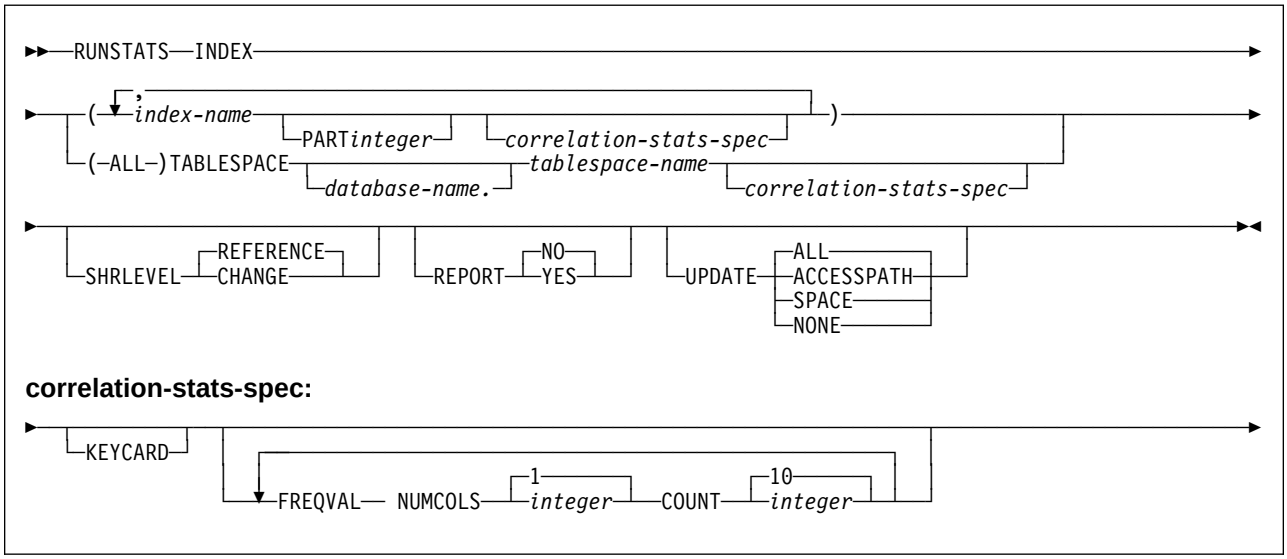
The **default** is **UPDATE ALL**.

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.

RUNSTATS INDEX Syntax



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 2-7.

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.

KEYCARD
Collect key cardinalities on the concatenation of key columns for the specified indexes.

FREQVAL
Controls the collection of frequent value statistics. If FREQVAL is specified, then it must be followed by two additional options:

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*.

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 **NO**.

YES
Indicates that the set of messages is output to SYSPRINT.

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 are updated in the catalog.

The **default** is **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 valid only when REPORT YES is specified.

Sample JCL and Control Statements

Examples

Example 1: Update Catalog Statistics While Allowing Changes: Update the catalog statistic columns for table space DSN8S51E and all its associated indexes. Permit other processes to make changes while this utility is executing.

```
//UTILSAMP EXEC PGM=DSNUTILB,REGION=1024K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*
//*********************************************************************
//*  DATA SETS USED BY THE UTILITY                                    *
//*********************************************************************
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *

 RUNSTATS TABLESPACE DSN8D51A.DSN8S51E
   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 DSN8S51E) while this utility is executing.

```
RUNSTATS INDEX (DSN8510.XEMPL1,DSN8510.XEMPL2)
```

Example 3: Update Index Statistics: Obtain statistics on the index XEMPL1.

```
RUNSTATS INDEX (DSN8510.XEMPL1)
```

Example 4: Update Statistics for Several Tables: Update the catalog statistics for all columns in the TCONA and TOPTVAL tables in table space DSN8D51P.DSN8S51C. Update the column statistics for the LINENO and DSPLINE columns in the TDSPTXT table in table space DSN8D51P.DSN8S51C.

```
RUNSTATS TABLESPACE(DSN8D51P.DSN8S51C) 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(DSN8D51P.DSN8S51C) 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 DSN8D51A.DSN8S51E
  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 DSN8D51A.DSN8S51E
  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 DSN8D51A.DSN8S51E
  REPORT YES
  UPDATE NONE
```

Example 9: Update Statistics For a Partition: Update the statistics for the table space and the partitioned index after a change to partition 1.

```
RUNSTATS TABLESPACE DSN8D51A.DSN8S51E PART 1 INDEX(DSN8510.XEMP1 PART 1)
```

Chapter 2-18. 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.

Syntax Diagram: For a diagram of STOSPACE syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 2-301.

Output: The output from STOSPACE consists of new values in a number of catalog tables. See “Reviewing STOSPACE Output” on page 2-300 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.

Instructions for Running STOSPACE

In order to run STOSPACE, you must:

1. Prepare the necessary data sets, as described in “Creating JCL Statements for Required Data Sets” on page 2-298.
2. Create JCL statements, by using one of the methods described in “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7. (For examples of JCL for STOSPACE, see “Sample JCL and Control Statement” on page 2-302.)
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 2-298. (For a complete description of the syntax and options for STOSPACE, see “Syntax and Options of the Control Statement” on page 2-301.)
4. Check the compatibility rules in “Concurrency and Compatibility” on page 2-301 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 2-301.

Invoking STOSPACE

See “Chapter 2-1. Invoking DB2 Online Utilities” on page 2-7 for an explanation of ways to invoke DB2 utilities.

Phases and Data Flow

The STOSPACE utility operates in these phases:

Phase	Description
UTILINIT	Initialization and setup
STOSPACE	
UTILTERM	Cleanup

Creating JCL Statements for Required Data Sets

Table 65 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 65. 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 objects are named in the utility control statement and do not require DD cards 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 2-301 for STOSPACE syntax and option descriptions. See "Sample JCL and Control Statement" on page 2-302 for examples of STOSPACE usage.

Instructions for Specific Tasks

For the tasks described, specify the options and values shown with your utility control statement. The following tasks are described here:

"Ensuring Availability of Objects Needed by STOSPACE"
"Obtaining Statistical Information with STOSPACE"
"Understanding the Values in a SPACE column" on page 2-300

Ensuring Availability of Objects Needed 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 needed. However, in order 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 66 lists statistical information recorded by the STOSPACE utility that is useful for space allocation decisions.

Table 66 (Page 1 of 2). 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

Table 66 (Page 2 of 2). DB2 Catalog Data Collected by STOSPACE

Catalog Table	Column Name	Column Description
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

Note: The STOSPACE utility does not collect data for objects belonging to databases defined as ROSHARE READ.

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 DSN8S51E table space in the DSN8D51A database, first run the STOSPACE utility, and then execute this SQL statement:

General-use Programming Interface

```
SELECT SPACE
  FROM SYSIBM.SYSTABLESPACE
  WHERE NAME = 'DSN8S51E
  AND DBNAME = 'DSN8D51A';
```

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 DSN8G510, as well as SYSIBM.SYSTABLESPACE and SYSIBM.SYSINDEXES for every table space and index belonging to DSN8G510, use the following utility:

```
STOSPACE STOGROUP DSN8G510
```

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. 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 needed. However, in order 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.

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.

Considerations for Running STOSPACE

For user-defined spaces, STOSPACE does not record any statistics.

Space statistics for shared read-only objects are not gathered or inserted into the DB2 catalog.

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 2-28.

Concurrency and Compatibility

STOSPACE does not set a utility restrictive state on the target object.

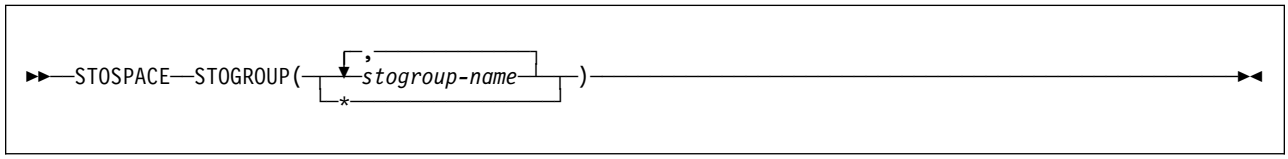
STOSPACE can run concurrently on the same target object with any utility. However, since STOSPACE updates the catalog, concurrent STOSPACE utility jobs or other concurrent appplications that update the catalog could cause timeouts and deadlocks.

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 1-3.



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 2-7.

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.

Sample JCL and Control Statement

Example

Example: Update Catalog SPACE Columns: Update the DB2 catalog SPACE columns for storage group DSN8G510.

```
//UTILSAMP EXEC PGM=DSNUTILB,REGION=1024K,
//             PARM='V51A,DSNTEX'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//*
//*********************************************************************
//*  DATA SETS USED BY THE UTILITY                                    *
//*********************************************************************
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
  STOSPACE STOGROUP DSN8G510
//*
```

Section 3. Stand-Alone Utilities

Chapter 3-1. Invoking Stand-Alone Utilities

Stand-Alone Utilities

This chapter contains procedures and guidelines for creating utility control statements and describes the procedures for invoking the stand-alone utilities.

Creating Utility Control Statements

Utility control statements define the function 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.

Following 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 fixed-length 80-character records, columns 73 through 80 are ignored.
- The records are concatenated before being parsed; hence, 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 *stand-alone utility names*:

DSNJU003	DSNJUOO4	DSN1CHKR	DSN1COMP
DSN1COPY	DSN1LOGP	DSN1PRNT	DSN1SDMP

  At least one blank character must follow the name.

  For the IFC Selective Dump Utility (DSN1SDMP), the SYSIN stream should start with START TRACE.
- Other syntactical constructs in the utility control statement describe options; they can be separated from each other by an arbitrary number of blanks.
- The SYSIN stream can contain multiple utility control statements.

The online utility name determines which options can follow it. More than one utility control statement can be specified in the SYSIN stream.

Options are typically described by an *option keyword*, followed by a *parameter*. The parameter value can be a keyword. Parameters' values are sometimes enclosed in parentheses, but are not generally required to be. The syntax diagrams for utility control statements show parentheses where they are required.

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:

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. Parameter values are usually be enclosed in parentheses.

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 new DSNJLOGF utility avoids this delay by preformatting the active log data sets before bringing them on-line to DB2.

Invoking DSNJLOGF

Run DSNJLOGF as an MVS job.

Sample JCL to Invoke DSNJLOGF

```
//JOBLIB DD  DSN=DSN510.SDSNLOAD,DISP=SHR
//STEP1    EXEC PGM=DSNJLOGF
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSUT1   DD DSN=DSNC510.LOGCOPY1.DS01,DISP=SHR
//STEP2    EXEC PGM=DSNJLOGF
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSUT1   DD DSN=DSNC510.LOGCOPY1.DS02,DISP=SHR
//STEP3    EXEC PGM=DSNJLOGF
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSUT1   DD DSN=DSNC510.LOGCOPY2.DS01,DISP=SHR
//STEP4    EXEC PGM=DSNJLOGF
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSUT1   DD DSN=DSNC510.LOGCOPY2.DS02,DISP=SHR
```

Required Data Sets

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 DSNJLOGF Output

```
DSNJ991I DSNJLOGF START OF LOG DATASET PREFORMAT FOR JOB LOGFRMT  STEP1
DSNJ992I DSNJLOGF LOG DATA SET NAME = DSNC510.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
- Supply passwords for archive logs and DB2 system data bases
- 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 or the highest offloaded RBA value.

Syntax Diagram: For a diagram of DSNJU003 syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 3-17.

Before Running DSNJU003

This section describes considerations you need to know before you run 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

To use this utility, the authorization ID of the job must have either the requisite RACF authorization or, if the BSDS is password protected, the appropriate VSAM password for the data set.

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 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, use:

```
NEWLOG DSNAME=DSNC510.LOGCOPY1.DS04,COPY1
NEWLOG DSNAME=DSNC510.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 could 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 use:

```
DELETE DSNAME=DSNC510.LOGCOPY1.DS01
DELETE DSNAME=DSNC510.LOGCOPY2.DS01
```

Recording: To record information about an existing active log data set in the BSDS, you might use:

```
NEWLOG DSNAME=DSNC510.LOGCOPY2.DS05,COPY2,STARTIME=19910212205198,
              ENDTIME=19910412205200,STARTRBA=43F8000,ENDRBA=65F3FFF
```

You might 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: One of the following procedures must be used when DB2 is inactive (down).

Use the following procedure if you can use the access method services REPRO command:

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 used to create the original active log data sets. See *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, use 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 change log inventory utility (DSNJU003) with the DELETE statement to delete all active logs except the current active logs from the BSDS.

5. Run the change log inventory utility with the NEWLOG statement to add the active logs that you just deleted to the BSDS. Do not specify an RBA range so that the logs are added as empty.
6. Start DB2.
7. 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.
8. Repeat steps 2 on page 3-10 through 6 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 could find it necessary to increase the size of your archive log data sets.

Making Changes for Archive Logs

Adding: When the recovery of an object depends upon 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 use:

```
NEWLOG DSNAME=DSNC510.ARCHLOG1.D89021.T2205197.A0000015,COPY1VOL=DSNV04,
UNIT=TAPE,STARTRBA=3A190000,ENDRBA=3A1F0FFF,CATALOG=NO
```

Deleting: To delete an entire archive log data set on one or more volumes, you might use:

```
DELETE DSNAME=DSNC510.ARCHLOG1.D89021.T2205197.A0000015,COPY1VOL=DSNV04
```

Changing a password: If you change the password of an existing archive log data set, change also the information in the BSDS, as follows:

1. List the BSDS, using the print log map.
2. Delete the entry for the archive log data set with the changed password, using DELETE.
3. Name the same data set as a new archive log data set. Use NEWLOG and give the new password, the starting and ending RBAs, and the volume serials (which can be found in the print log map output).

No such action is needed when you change the password on an *active* log data set, because its password is not recorded in the bootstrap data set.

To change the password for new archive log data sets, use:

```
ARCHIVE PASSWORD=password
```

To stop placing passwords on new archive log data sets, use:

```
ARCHIVE NOPASSWD
```

Creating a Conditional Restart Control Record

To create a new conditional restart control record in the BSDS, you invoke the change log inventory utility and use the CRESTART control statement. For example, to truncate the log, to specify the earliest log RBA, and to bypass backout, you use a statement similar to this.

```
CRESTART CREATE,STARTRBA=28894,ENDRBA=58000,BACKOUT=NO
```

To specify a cold start, you make the values of STARTRBA and ENDRBA equal. A statement like the following can be used.

```
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 can *not* use the other active log or the status is STOPPED , the problem must be fixed manually.
   a. Check to see if the data set has been off-loaded. 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 off-loaded, copy the data to a new VSAM data set. If the data set has been off-loaded, create a new VSAM data set to be used as an active log data set.
   c. Use DELETE to remove information about the bad data set from the BSDS.
   d. Use NEWLOG to specify 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.

PRINT LOG MAP should be used before and after running CHANGE LOG INVENTORY 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.LOCCOPYN.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 is used, pairs of data sets with equal mm values will usually be used together. For example, DSNC120.LOGCOPY1.DS02 and DSNC120.LOGCOPY2.DS02 would be used together.

However, after CHANGE LOG INVENTORY DELETE and NEWLOG, the primary and secondary series can become unsynchronized, even if the NEWLOG data set name 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 them both together with NEWLOG statements.

The data set themselves do not require deletion and redefinition.

To ensure consistent results from this utility, execute CHANGE LOG INVENTORY on the same MVS system that the DB2 online subsystem is executing under.

If misused, CHANGE LOG INVENTORY can compromise the viability and integrity of the DB2 system. This utility must only be used by a highly-skilled person such as the DB2 System Administrator and then only after careful consideration.

Before initiating a conditional or cold restart, you should consider making back up 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.

NEWLOG and DELETE Datasets

The NEWLOG and DELETE statements add and delete references to data sets in the BSDS. The log data set are not changed in any way. If DELETE and NEWLOG are used for an active log data set reference in the BSDS, the referenced log data set itself does not need alteration.

NEWCAT

NEWCAT defines the high-level qualifier used for the following:

- Catalog table spaces and index spaces
- Directory table spaces and index spaces

At startup, the DB2 system will check that the name recorded with NEWCAT in the BSDS is the high-level qualifiers of the DB2 system table spaces defined in the load module for subsystem parameters.

NEWCAT is normally used only at installation time. See "Renaming DB2 System Data Sets" for an additional function of NEWCAT.

```
//S2 EXEC PGM=DSNJU003
//SYSUT1   DD DSN=DSNC120.BSDS01,DISP=OLD
//SYSUT2   DD DSN=DSNC120.BSDS02,DISP=OLD
//SYSPRINT DD SYSOUT=*
  NEWCAT VSAMCAT=DBP1
```

Figure 22. Output Produced When Changing High-Level Qualifier

```
  NEWCAT VSAMCAT=DBP1
 DSNJ210I OLD VASAM CATALOG NAME=DSNC120 NAME=DBP1
 DSNJ225I NEWCAT OPERATION COMPLETED SUCCESSFULLY
 DSNJ200I DSNJU003 CHANGE LOG INVENTORY UTILITY
          PROCESSING COMPLETED SUCCESSFULLY
```

Figure 23. Output Produced When Changing High-Level Qualifier

Renaming DB2 System Datasets

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 to be prepared for operational errors.
3. Execute CHANGE LOG INVENTORY 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 data base.
9. Optionally ALTER table spaces in DSNDB04 and user data bases.

Renaming DB2 Active Log Datasets

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 to be prepared for operational errors.
3. Delete the reusable active log data sets with IDCAMS, but keep the currently active log.
4. Define a new set of active log data sets with IDCAMS.
5. Execute CHANGE LOG INVENTORY to remove deleted active log data set names and to define the new active log data set names in the BSDS.
6. Start and use DB2 normally.

7. When the currently active log is archived and become reusable, it can be deleted.

Renaming DB2 Archive Log Datasets

Archive log data sets need not be renamed, because:

- Old archive logs are replaced using the normal maintenance cycle.
- RECOVER 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.

Creating the DSNJU003 Control Statement

See “Syntax and Options of the Control Statement” on page 3-17 for DSNJU003 syntax, option descriptions, and example control statements.

Change log inventory provides the following statements:

- NEWLOG
- DELETE
- ARCHIVE
- SYSTEMDB
- CRESTART
- NEWCAT
- DDF
- CHECKPT
- HIGHRBA

You can use more than one statement of each type. 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.

Multiple Statement Operation

When executing 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 will not occur for any operation specified in the statement in error and any subsequent statements.

Including the Required JCL

Utility Invocation

The following statement invoking the utility can be included only in a batch job:

```
//EXEC   PGM=DSNJU003
```

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. The 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
Is required to specify and allocate the bootstrap data set.

SYSUT2
Specifies and allocates a second copy of the bootstrap data set. The statement is required if you use dual BSDSs.

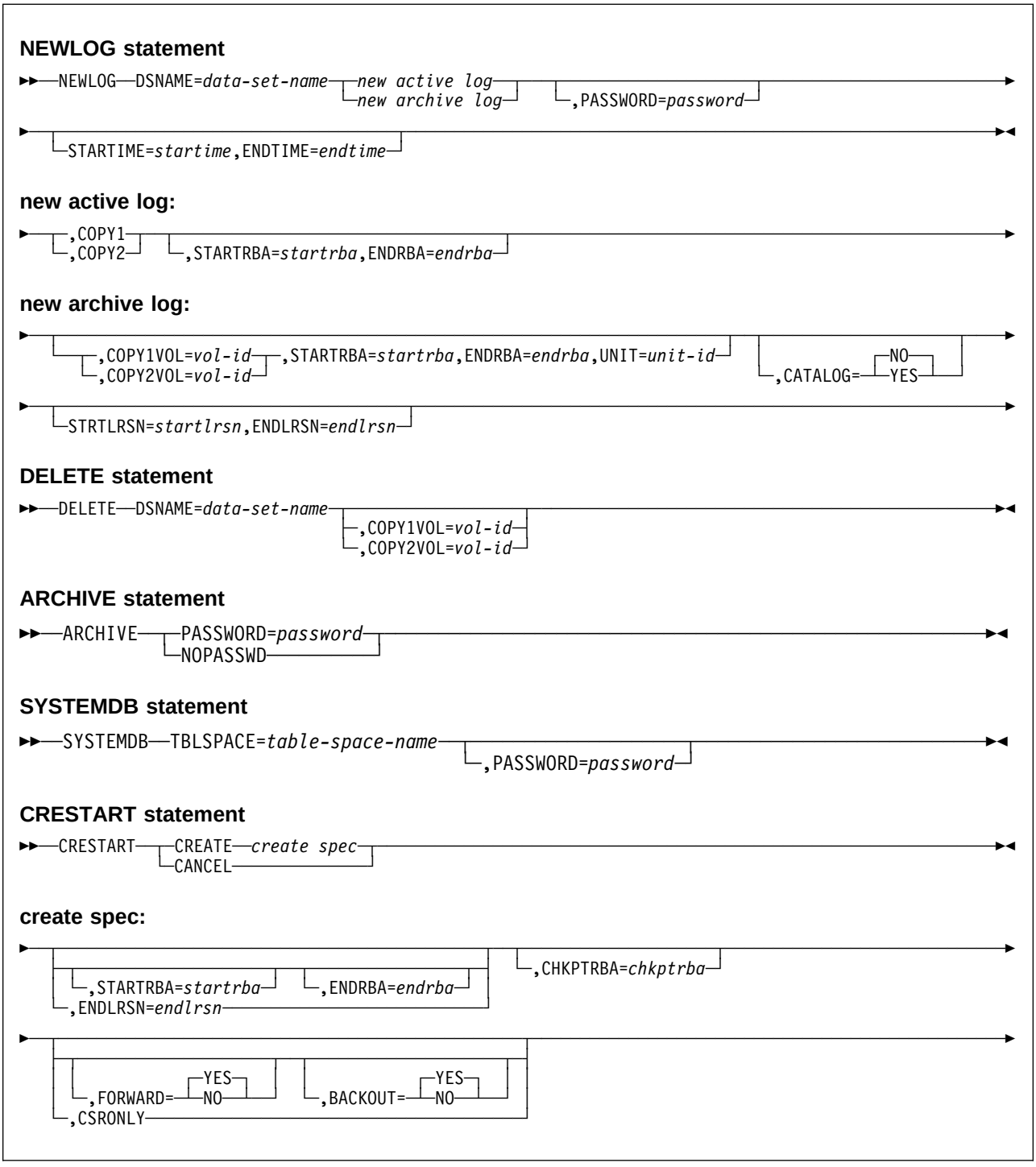
Dual BSDSs and DSNJU003: With each execution of DSNJU003, the BSDS timestamp field is updated with the current system time. If you run DSNJU003 separately for each copy of a dual copy BSDS, the timestamp fields are not synchronized, and DB2 fails at startup. If you changed the contents of the BSDS copy by running DSNJU003, then DB2 issues error message DSNJ122I. Therefore, if DSNJU003 is used to update dual copy BSDSs, both BSDSs must be updated within a single execution of DSNJU003.

SYSPRINT
Is required to specify a data set for print output. The logical record length (LRECL) is 125.

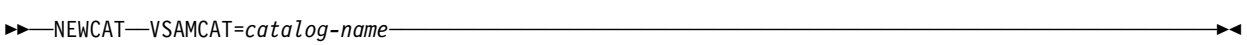
SYSIN
Is required to specify the input data set for statements. The logical record length (LRECL) is 80.

Syntax and Options of the Control Statement

DSNJU003 (Change Log Inventory) Syntax

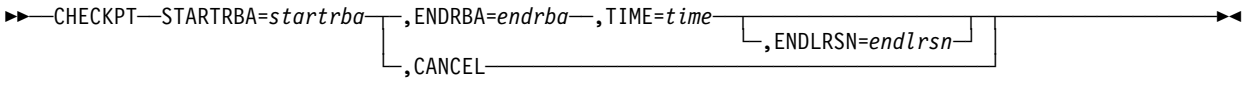


NEWCAT statement

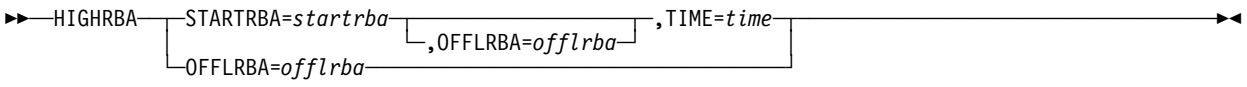


DDF statement

CHECKPT statement



HIGHRBA statement



Option Descriptions

The following statements tell what operations are to be performed. Their keywords and parameters describe the options to be used.

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, COPY2, and PASSWORD=.

- An active log data set that is replacing one that encountered an I/O error.

  Use only the keywords DSNAME=, COPY1, COPY2, STARTRBA=, ENDRBA=, and PASSWORD=.

- An archive log data set volume.

  Use only the keywords DSNAME= ,COPY1VOL=, COPY2VOL=, STARTRBA=, ENDRBA=, UNIT=, CATALOG=, PASSWORD=, STRTLRSN=, and ENDLRSN=.

  If you create an archive log data set and add it to the BSDS with this utility, you can select 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 select a name using the same naming convention as DB2, then 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 off-loads to it the next time it archives. (The active log that previously was not off-loaded is off-loaded to this data set.)

The new active logs defined cannot specify a start and end LRSN. The start and end LRSN for new active logs which contain active log data will be read at DB2 start up time from the new active log data sets specified in the Change Log Inventory NEWLOG statement(s). For new archive logs defined with Change Log Inventory, the user must specify the start/end RBAs and for data sharing, also the start and end LRSNs. DB2 start up will 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.

ARCHIVE
Gives a 1- to 8- character password for *all* archive data sets created after the archive operation. The password is added to the installation's MVS password data set every time a new archive log data set is created. The NOPASSWD parameter removes the password protection for all archives created after the archive operation. No other keywords can be used with NOPASSWD.

SYSTEMDB
Gives VSAM passwords for the data sets that support the following DB2 databases:

DSNDB01 (the DB2 directory)
DSNDB06 (the DB2 catalog).

CRESTART
Controls the next restart of DB2, either by creating a new conditional restart control record or by canceling the one currently active.

Attention: This statement can override DB2's efforts to maintain data in a consistent state. Do not use the statement without understanding “conditional restart,” which is described in Section 4 (Volume 1) of *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 once 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 "conditional restart" and "checkpoint processing," which are described in Section 4 (Volume 1) of *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 "conditional restart" which is described in Section 4 (Volume 1) of *Administration Guide*.

Option Descriptions

DSNAME=*data-set-name*
Specifies a log data set.

data-set-name can be up to 44 characters long.

PASSWORD=*password*
The passwords for the NEWLOG, ARCHIVE, and SYSTEMDB statements are data set passwords, and must follow standard VSAM convention: 1 to 8 alphanumeric characters (A-Z, 1-9) or special characters (& * + - .; ' /).

The DDF password follows VTAM convention, but DB2 restricts it to 1 to 8 alphanumeric characters. The first character must be either a capital letter or an alphabetic extender. The rest can consist of alphanumeric and alphabetic extenders.

In the NEWLOG statement, *password* assigns a password to the data set. It is stored in the BSDS and used in any subsequent access to archive log data sets.

In the ARCHIVE statement, *password* assigns a password to the archive log data sets.

In the SYSTEMDB statement, *password* assigns a password to the directory (DSNDB01) and the catalog (DSNDB06).

In the DDF statement, *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 used to define DB2 to VTAM.

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:

yyyyddhhmmsst, 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, then trailing zeros are added.

If STARTIME is specified, then 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 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*
On the NEWLOG statement, *startrba* gives the log RBA (relative byte address within the log) of the beginning of the replacement active log data set or the archive log data set volume specified by DSNAME. *startrba* is a hexadecimal number of up to 12 digits. If you use fewer than 12 digits, leading zeros are added. *startrba* must end with '000' or DB2 will return a DSNJ4381 error message. The RBA can be obtained from messages or by printing the log map.

On the CRESTART statement, *startrba* is the earliest RBA of the log to be used during restart. If you omit the option, 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*
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. *endrba* is a hexadecimal number of up to 12 digits. If you use fewer than 12 digits, leading zeros are added. *endrba* must end with 'FFF' or DB2 will return a DSNJ4381 error message.

On the CRESTART statement, *endrba* is the last RBA of the log to be used during restart, and 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 RBA specified will be the beginning RBA of the new log.

On the CHECKPT statement, *endrba* indicates the end checkpoint log record corresponding to the start checkpoint log record.

For the *endrba* format, see the NEWLOG statement.

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 digits. If you use fewer than 12 digits, leading zeros are added. In a data sharing environment, run Print Log Map to find an archive log data set and start and end RBAs and LRSNs.

ENDLRSN=*endlrsn*

On the NEWLOG statement, *endlrsn* is the LRSN in log record header of the last log record on the new archive data set. *endlrsn* is a hexadecimal number of up to 12 digits. If you use fewer than 12 digits, leading zeros are added. In a data sharing environment, run Print Log Map to find an archive log data set and start and end RBAs and LRSNs.

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 only valid 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.

COPY1VOL=*vol-id*

vol-id is the volume serial of the copy-1 archive log data set specified after DSNAME.

COPY2VOL=*vol-id*

vol-id is the volume serial of the copy-2 archive log data set specified after DSNAME.

UNIT=*unit-id*

unit-id is the device type of the archive log data set named after DSNAME.

CATALOG

Tells whether the archive log data set is cataloged.

NO

Tells 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.

The **default** is **CATALOG NO**.

YES

Tells 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.

TBLSPACE=*table-space-name*
Gives the table space or index space qualifier for the data set to be assigned a VSAM password or from which the current VSAM password is to be removed. The qualifier is found in the data set name, which has this format:

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

where:

catname	VSAM catalog name or alias
x	C or D
dbname	database name
tsname	table space name
nnn	data set integer

Use the SYSTEMDB statement only if the database name (*dbname*) is DSNDB01 or DSNDB06. Then use the table space name (*tsname*) for *table space*.

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 makes the currently active conditional restart control record inactive. 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 containing a starting RBA which matches the parameter specified by the STARTRBA keyword.

CHKPTRBA=*chkptrba*
Is the log RBA (relative byte address within the log) of the start of the checkpoint record to be used during restart.

If you use STARTRBA or ENDRBA, and 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=
Tells 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.

The **default** is **FORWARD=YES**.

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=
Tells 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.

The **default** is **BACKOUT=YES**.

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 made inactive. 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 8 characters long. The first character must be alphabetic, while 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.

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, where zero indicates that DDF's TCP/IP support is being 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 2–phase commit resynchronization requests. This value must be a decimal number between 0 and 65534, where zero indicates that DDF's TCP/IP support is being 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 tho 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 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.

OFFLRBA=*offlrba*
Specifies the highest off-loaded RBA in the archive log.

offlrba is a hexadecimal number of up to 12 digits. If you use fewer than 12 digits, leading zeros are added. The value must end with hexadecimal `'FFF'`.

Sample Control Statements

Examples

Example 1: In the following example, the first statement adds a new archive log data set. The second statement establishes a new password for subsequent archive data sets:

```
NEWLOG DSNAME=DSNREPAL.A0001187,COPY1VOL=DSNV04,UNIT=SYSDA,
STARTRBA=3A190000,ENDRBA=3A1F0000,CATALOG=NO,PASSWORD=SYSNKZX
ARCHIVE PASSWORD=SYSNLZX
```

Example 2: The following statement deletes a data set:

```
DELETE DSNAME=DSNREPAL.A0001187,COPY1VOL=DSNV04
```

Example 3: 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: The following statement adds 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 copy 1 and copy 2 of all active and archive log data sets
- Passwords for those data sets, if provided
- Active log data sets 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.

Further information regarding the DSNJU004 utility appears in Section 4 (Volume 1) of *Administration Guide*.

Before Running DSNJU004

This section describes considerations you need to know before you run DSNJU004.

Environment

The DSNJU004 program runs as a batch job.

This utility can be executed when DB2 is both 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 control of the same MVS system.

Authorization Required

To use this utility, the user ID of the job must have requisite RACF authorization or, if the BSDS is password protected, the appropriate VSAM password for the data set.

Recommendations

- For dual BSDSs, execute print log map 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 that the DB2 online subsystem is executing under.
- Regular, possibly daily, execution is advisable to keep a record of recovery log data set usage.
- Use print log map to document changes made by the change log inventory utility.
- Restrict print log map usage to the DB2 system administrator, because it shows DB2 data set passwords.

Creating the JCL to Run DSNJU004 (Print Log Map)

The following EXEC statement is used to invoke this utility:

```
//   EXEC   PGM=DSNJU004
```

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. The 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

Required to specify and allocate the bootstrap data set. 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

Required to specify a data set or print spool class for print output. The logical record length (LRECL) is 125.

SYSIN (optional)

Contains the control statement. If you do not specify the SYSIN DD statement, then 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 use 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 any of these options:

MEMBER *
MEMBER (*member-name)*

MxxBSDS

Names the BSDS data set of a group member whose information is to be listed. There must be one such DD statement for each member. The statements are required if the control statement specifies MEMBER DDNAME. *xx* represents a 2-digit number. You must use consecutive 2-digit numbers from 01 to the total number of members required. If there is a break in the sequence of numbers, any number after the break is ignored.

Sample JCL

Example:

```
//PLM      EXEC PGM=DSNJU004
//SYSUT1   DD DSN=DBD1.BSDS01,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
           MEMBER *
```

Reviewing DSNJU004 (Print Log Map) Output

Figure 24 on page 3-32 shows an example of the print log map output, with the following information:

- The data set name (DSN) of the BSDS.
- The system date and time (SYSTEM TIMESTAMP) set during the stopping of the subsystem.
- 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 figure is updated each time the log buffers are physically written to DASD.
- The highest RBA off-loaded.
- 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), password, 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), password, 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 print log map, see “Reading Conditional Restart Control Records” on page 3-34.
- The contents of the checkpoint description queue. For a description of this output, see Figure 25 on page 3-34.
- The distributed data facility (DDF) communication record. This record contains the DB2-defined location name, the VTAM-defined LUNAME, and the password used to connect DB2 to VTAM. 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 24 on page 3-32 shows an example of the print log map 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 could not be updated for several seconds, minutes, or even hours.
- When, due to an error, DB2 drops into a 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:

Conditional restart control record (heading)
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:

Checkpoint queue (heading)
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 five values listed in Table 67. For an example of how the status appears in print log map output, see Figure 24 on page 3-32.

Table 67. 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 off-loaded. 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 off-loaded.
STOPPED	The off-load processor encountered an error while reading a record, and that record could not be obtained from the other copy of the active log. See Section 4 (Volume 1) of *Administration Guide.*
TRUNCATED	One of these cases: • An I/O error occurred, and DB2 has stopped writing to this data set. The active log data set is off-loaded, 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 command -ARCHIVE LOG was issued while this data set was the current active log data set.

```
*************************************************************************************************************
*                                                                                                           *
*              LOG MAP OF THE BSDS DATA SET BELONGING TO MEMBER 'V51A    ' 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=DSNC510.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=DSNC510
         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:            V51A
           MEMBER ID:                 1
           GROUP NAME:                DSNCAT
           BSDS COPY 1 DATA SET NAME: DSNC510.BSDS01
           BSDS COPY 2 DATA SET NAME: DSNC510.BSDS02
         MEMBER NAME:                 V51B
           MEMBER ID:                 2
           GROUP NAME:                DSNCAT
           BSDS COPY 1 DATA SET NAME: DSNC510.BSDS01
           BSDS COPY 2 DATA SET NAME: DSNC510.BSDS02
```

Figure 24 (Part 1 of 2). Sample Print Log Map Utility Output

```
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=DSNC510.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=DSNC510.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=DSNC510.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=DSNC510.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=DSNC510.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=DSNC510.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
                         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 24 (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 saved in the BSDS. These printed values will 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 25.

```
                   CHECKPOINT QUEUE
                 09:38:58 AUGUST 28, 1987
TIME OF CHECKPOINT       09:38:35 AUGUST 28, 1987
BEGIN CHECKPOINT RBA             000000047C10
END CHECKPOINT RBA               000000048510
TIME OF CHECKPOINT       09:23:55 AUGUST 28, 1987
BEGIN CHECKPOINT RBA             000000035010
END CHECKPOINT RBA               000000035780
TIME OF CHECKPOINT       09:08:32 AUGUST 28, 1987
BEGIN CHECKPOINT RBA             000000029000
END CHECKPOINT RBA               0000000297A0
```

Figure 25. Sample Print Log Map Description of Checkpoints

A sample description of a conditional restart control record is shown in Figure 26 on page 3-35.

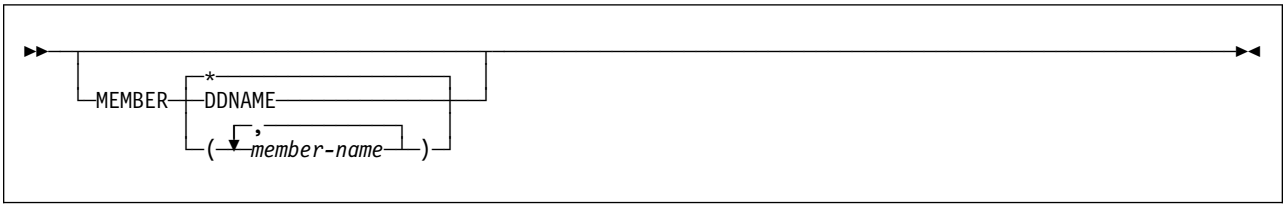
```
CRCR IDENTIFIER  0001
     USE COUNT   1
     RECORD STATUS
         CRCR NOT ACTIVE
         SUCCESSFUL RESTART
     PROCESSING STATUS
         FORWARD = YES
         BACKOUT = NO
     STARTRBA                          000000028894
     ENDRBA                            000000058000
     EARLIEST REQUESTED RBA            000000027B00
     FIRST LOG RECORD RBA              0000000288B0
     ORIGINAL CHECKPOINT RBA           00000005A390
     NEW CHECKPOINT RBA (CHKPTRBA)     000000047C10
     END CHECKPOINT RBA                000000048510
     CRCR CREATED              10:25:02 AUGUST 28, 1987
     BEGIN RESTART             10:30:30 AUGUST 28, 1987
     END RESTART               10:35:42 AUGUST 28, 1987
     TIME OF CHECKPOINT        09:38:35 AUGUST 28, 1987
     RESTART PROGRESS                  STARTED     ENDED
                                       =======     =====
         CURRENT STATUS REBUILD          YES         YES
         FORWARD RECOVERY PHASE          YES         YES
         BACKOUT RECOVERY PHASE          YES         YES
```

Figure 26. Sample Print Log Map Description of a CRCR

Syntax and Options of the Control Statement

Using the SYSIN data set allows you to list information from any or all BSDSs of a data sharing group.

Control Statement Syntax



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 member(s) named.

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 detect any damage to the catalog and directory as soon as possible.

Syntax Diagram: For a diagram of DSN1CHKR syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 3-41.

Before Running 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.

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, then it can be copied into a VSAM data set with DSN1COPY and checked 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.

It is a good idea to 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 DSNICHKR only on a valid table space.

Do not run DSN1CHKR on table spaces DSNDB06.SYSCOPY, DSNDB06.SYSGPAUT, DSNDB06.SYSPKAGE, DSNDB06.SYSSTATS, DSNDB06.SYSSTR, DSNDB06.SYSUSER, DSNDB01.SCT02, DSNDB01.SPT01, DSNDB01.SYSLGRNX, DSNDB01.SYSUTILX, and DSNDB06.SYSDDF.

Authorization Required

None is required. However, if any of the data sets are RACF protected, the authorization ID of the job must have the necessary RACF authority. Also, if the data sets are password protected, the authorization ID of the process must have the appropriate VSAM password to execute this utility.

Creating the JCL to Run DSN1CHKR

See “Syntax and Options of the Control Statement” on page 3-41 for DSN1CHKR syntax and option descriptions.

Required Data Sets

DSN1CHKR uses 2 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 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.

Sample JCL for Running DSN1CHKR

Example 1: In the example shown in Figure 27 on page 3-39, DSN1CHKR is run 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. Once 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. The first pointer is located on page 2, at an offset of 6 bytes from record 1. The second pointer is located on page B, at an offset of 6 bytes from record 1.

The RIDs in Step 5 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(SSTR)
     -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
/*
//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(SSTR)
     -START DB(DSNDB06) SPACENAM(SYSDBASE)
 END
/*
```

Figure 27 (Part 1 of 2). Sample JCL for running DSN1CHKR on a temporary data set

```
//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 27 (Part 2 of 2). Sample JCL for running DSN1CHKR on a temporary data set

Example 2: In the example shown in Figure 28, DSN1CHKR is run on the 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(SSTR)
     -STOP DB(DSNDB06) SPACENAM(SYSDBASE)
 END
/*
//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(SSTR)
     -START DB(DSNDB06) SPACENAM(SYSDBASE)
 END
/*
```

Figure 28. Sample JCL for running DSN1CHKR on a stopped table space.

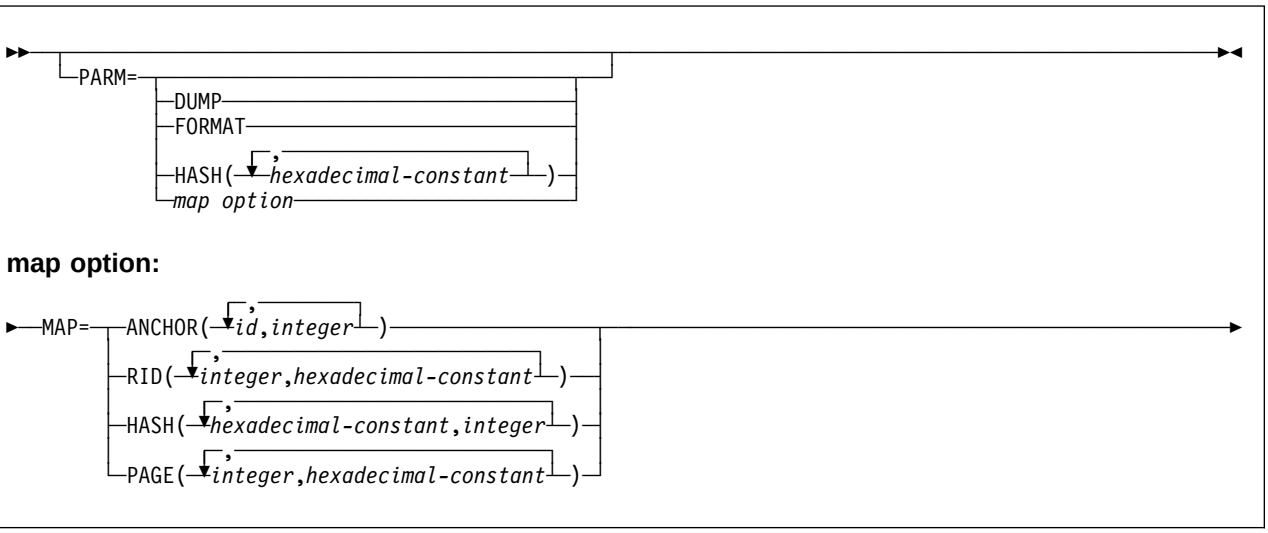
After Running DSN1CHKR

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.

Syntax and Options of the Control Statement

DSN1CHKR Syntax



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 using commas but no blanks. If you do not specify any parameters, DSN1CHKR scans all table space pages for broken links and 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.

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 *Administration Guide*.

This utility can be run on the following types of data sets containing uncompressed data:

- DB2 image copy data sets (full or incremental)
- 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 index spaces.

Syntax Diagram: For a diagram of DSN1COMP syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 3-46.

Considerations for Running DSN1COMP

This section describes considerations to keep in mind when 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, be sure that the DB2 data sets that are to be used are not currently allocated to DB2, by issuing the DB2 STOP DATABASE command.

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 are RACF-protected, the authorization ID of the job must have RACF authority. Also, if the data sets are password protected, the authorization ID of the job must have the appropriate VSAM passwords to execute this 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 REORG (the default invocation).

Without the REORG option, DSN1COMP uses the first *n* rows to fill the 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, there will be very little savings. With the REORG option, DSN1COMP processes all the rows, including those used to build the dictionary, which produces a greater compression savings estimate.

Running DSN1COMP on Incremental Image Copies

If the header page is not included in the incremental image copy, the DBID and PSID are zero. The missing header page can have an adverse effect on compression estimates for incremental image copies of segmented table spaces, because DSN1COMP does not know the size of the segments; as a result, the dictionary pages required are not a multiple of the segment size. However, the number of dictionary pages is usually small compared to the number of compressed pages, and the effect on the compression report is negligible.

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 number for “noncmppages” than the 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.

Creating the JCL to Run DSN1COMP

See “Syntax and Options of the Control Statement” on page 3-46 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. That data set can be a sequential data set or a VSAM data set.

Disposition for this data set must be specified as OLD (DISP=OLD) to ensure that it is not in use by DB2. 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 or index space that is larger than 2 gigabytes, or is a partitioned table or index space, you must ensure the correct data set is specified.

Sample JCL and Control Statements

Example 1: Sample JCL for 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: Sample DSN1COMP JCL 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=DSNC510.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=DSNC510.DSNDBD.DB254SP4.TS254SP4.I0001.A001,DISP=SHR
/*
//
```

Reviewing DSN1COMP Output

Message DSN1941

If you receive this message, use a data set with more rows as input, or specify a larger ROWLIMIT.

Sample DSN1COMP Report

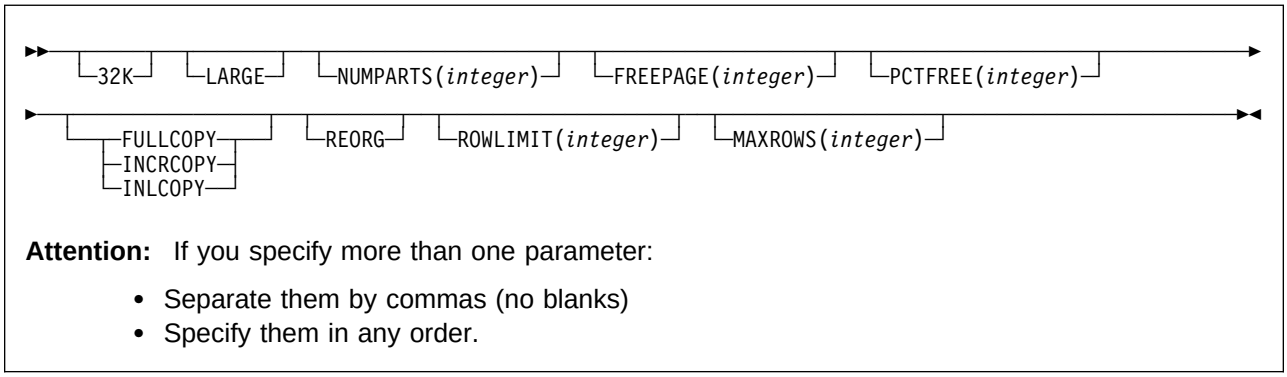
Figure 29 on page 3-46 shows a sample of the output that DSN1COMP generates.

```
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 29. Sample DSN1COMP Report

Syntax and Options of the Control Statement

DSN1COMP Syntax



Attention: 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 4KB.

LARGE
Specifies that the input dataset has been defined as LARGE partitioned table space. If the LARGE keyword is omitted for a LARGE partitioned table space or specified for a table space which is not LARGE, DSN1COMP may produce unpredictable results.

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 dataset is for a LARGE partitioned table space, 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 DSN1946 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 command 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 command 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 miscellaneous error messages or unpredictable results. If this data is partitioned, also specify the NUMPARTS parameter to identify the number of partitions.

INCRCOPY

Specifies that an incremental image copy of the data is used as input. Omitting this parameter when the input is an incremental image copy can cause miscellaneous error messages or unpredictable results. If the data is partitioned, also specify the NUMPARTS parameter to identify the number of partitions.

INLCOPY

Specifies that the input data is an inline copy data set.

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 required by DSN1COMP. 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 10MB of the table space.

MAXROWS(*integer***)**

Specifies the maximum number of rows that DSN1COMP will 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 command CREATE TABLESPACE or ALTER TABLESPACE.

Chapter 3-7. DSN1COPY

The DSN1COPY stand-alone utility allows you to 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 the log RBA recorded in each index or data page to 0.

- 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. DSN1COPY abends.
- If you require only a printed hexadecimal dump of a data set, use DSN1PRNT rather than DSN1COPY.

Syntax Diagram: For a diagram of DSN1COPY syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 3-58.

Restrictions on use of 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, data definitions for the source and target table spaces, tables, and indexes must be identical; otherwise, unpredictable results can occur.

Before Using DSN1COPY

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.

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, data definitions for the source and target table spaces, tables, and indexes must be identical; otherwise, unpredictable results can occur.

Environment

DSN1COPY is executed as an MVS job, and could be executed when the DB2 subsystem is either active or not active.

If you execute DSN1COPY when DB2 is active, follow the guidelines below.

1. Start the table space as read-only using -START DATABASE.

2. Issue the QUIESCE utility with the WRITE (YES) option to externalize all data 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 are RACF-protected, the authorization ID of the job must have the necessary RACF authority. Also, if the data sets are password protected, the authorization ID of the job must have the appropriate VSAM passwords to execute this utility.

The SYSUT1 data set can be:

- 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 the commands below should be executed before running DSN1COPY:

```
-DISPLAY DATABASE (database_name) SPACENAM(space_name) RESTRICT
-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 made up of a number of data sets, ensure that the correct data set is specified. For example, if a CHECK on the pages of the second partition of a partitioned table space is to be made, then the second data set of the table space has to be coded for SYSUT1.

Defining the Output Dataset

The SYSUT2 data set can be:

- A sequential data set
- A DB2 table space data set
- A DB2 index space data set
- DUMMY.

Specify a DUMMY SYSUT2 DD card if 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:

- Use DSN1COPY to copy your existing data set to a sequential data set. Specify this data set as SYSUT1. Next, if your existing data set was defined

without the REUSE parameter, delete and redefine the data set. Specify your existing data set as SYSUT2.

- Use your existing DB2 data set as the SYSUT1 specification, creating a new VSAM data set for SYSUT2. 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:

- 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 use the NUMPARTS parameter, as it 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, then do the following:

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.

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

You must use the CHECK option when critical data is involved 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 there are no inconsistencies between data and indexes on the 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 and you do not specify the OBIDXLAT parameter, you must 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 re-created after the data DSN1COPY saved was created and before it was used
- When there is a difference in 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, you must 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.

After using the FULLCOPY parameter to restore a table space, you must recover any indexes associated with that table space. You can do this by using the RECOVER INDEX utility. For more information about the RECOVER INDEX utility, refer to “Chapter 2-12. RECOVER INDEX” on page 2-165.

Resetting Page Log RBA

The RESET option resets the log RBAs recorded in a table space or index space to 0 and CHECK processing is carried out whether it is explicitly requested or not.

| Do not use RESET 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(s), the following SYSUT2 data sets should be specified.

- If SYSUT1 is an image of a single partition, SYSUT2 should be the name of the FIRST data set of the table space. DSN1COPY will determine the correct target data set. Code the NUMPARTS(*nn*) parameter when *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 will allocate all of the target data sets. However, the target data sets must be previously defined using IDCAMS. The NUMPARTS parameter must be coded as above, because the table space is partitioned.
- 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 use NUMPARTS as this 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 will allocate 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 DSN1COPY, there are two ways to restore the indexes. The easier and safer option is to use the RECOVER INDEX utility. This will reconstruct the indexes from the data. However, for table spaces with millions of rows, RECOVER INDEX will take a long time. The alternative is to use DSN1COPY on the indexes. If the OBIDXLAT option was used for the data, then it must also be used for the indexes. Also, the indexes must have been copied at the same time as the data; otherwise, inconsistencies may be present.

Restoring Table Spaces With DSN1COPY

It is not possible to RECOVER TOCOPY to an image copy data set which is not referred to in a SYSIBM.SYSCOPY row for that table space or data set. An attempt to do so results in a "TOCOPY DATASET NOT FOUND" message.

The SYSIBM.SYSCOPY row might have been removed by MODIFY. If this has happened, and the image copy is a full image copy with SHRLEVEL REFERENCE, the table space or data set can be restored with DSN1COPY.

It is also possible to restore to an incremental image copy with DSN1COPY but for data integrity, it is necessary first to have restored the previous full image copy and any intermediate incremental image copies. It is the your responsibility to get the sequence of image copies right. DB2 can not help in this instance.

If DSN1COPY is used for point-in-time recovery, the table space becomes not recoverable. Because DSN1COPY executed outside of DB2's control, DB2 is not aware of this. 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 possible 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 or index space that is larger than 2 gigabytes or is a partitioned table space or index, you must ensure that the correct data set is specified. For example, to print a page range in the second partition of a 4 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 recover a table space), specify a DUMMY SYSUT2 DD card and specify the FULLCOPY parameter.

Creating the JCL to Run DSN1COPY

See “Syntax and Options of the Control Statement” on page 3-58 for DSN1COPY syntax and option descriptions.

Required Data Sets

Input data set Input to DSN1COPY. The DD name is SYSUT1.

Output data set Optional, output from DSN1COPY. The DD name is SYSUT2.

Message data set Data set for output messages. The DD name is SYSPRINT.

OBIDXLAT data set Data set defining 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 data set specified. If the input data set is a partitioned table space or index, ensure that you specify the NUMPARTS parameter and the correct data set. For example, to print a page range in the second partition of a 4 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.

Do not code unit and volume serial parameters when using VSAM data sets so that necessary information can be obtained from the integrated catalog facility catalog.

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 less than 65536 and 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. The second record contains the source and target PSIDs or ISOBIDs for indexes. All subsequent records in the SYSXLAT data set are for table OBIDs. Only the first two records are required for a type 1 index space; for a type 2 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 type 2 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 type 1 index spaces:

```
SELECT DBID, ISOBID FROM SYSIBM.SYSINDEXES
   WHERE NAME='index_name'
     AND CREATOR='creator_name';
```

For type 2 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 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

Sample JCL and Control Statements

Example 1: Sample JCL for Running DSN1COPY:

```
//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: Using the OBIDXLAT Parameter with the SYSXLAT DD Card:

```
//EXECUTE EXEC PGM=DSN1COPY,PARM='OBIDXLAT'
//STEPLIB  DD  DSN=PDS CONTAINING DSN1COPY
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  DSN=DSNC510.DSNDBC.DSN8D51P.DSN8S51C.I0001.A001,
//             DISP=OLD
//SYSUT2   DD  DSN=DSNC518.DSNDBC.DSN8D51P.DSN8S51C.I0001.A001,
//             DISP=OLD
//SYSXLAT  DD  *
260,280
2,10
3,55
6,56
7,57
/*
```

Example 3: Printing a Page:

```
 //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: Print 16 pages of a Non-Partitioned 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
```

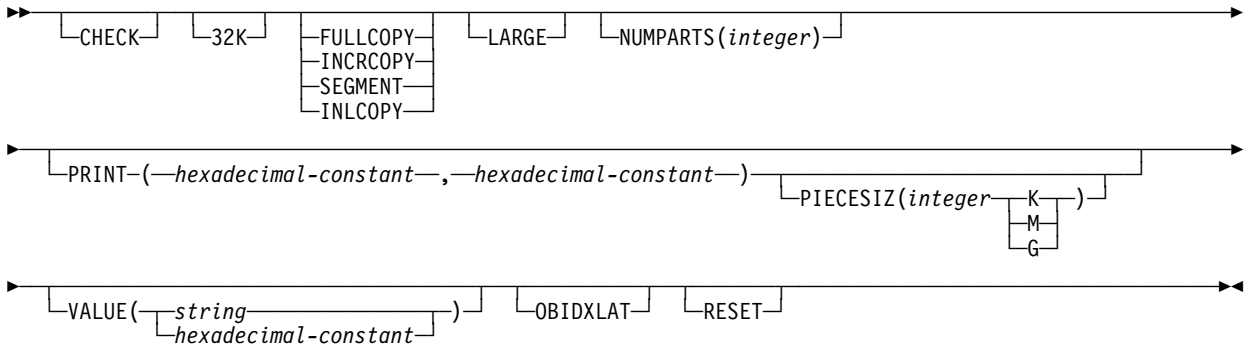
After Running DSN1COPY

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.

Syntax and Options of the Control Statement

DSN1COPY Syntax



Attention: 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 4KB.

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, 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 miscellaneous error messages or unpredictable results.

The FULLCOPY parameter requires SYSUT2 (output data set) to be either a DB2 VSAM data set or a "DUMMY".

INCRCOPY
Specifies that an incremental image copy of the data is used as input. DSN1COPY with the INCRCOPY parameter does updates to existing data sets so redefinition of the data sets must not be done. 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.

INLCOPY
Specifies that the input data is an inline copy data set.

LARGE
Specifies that the input data set has been defined as a large partitioned table space or an index data set backed by a large partitioned table space. If the LARGE option is omitted for a large partitioned table space or specified for a table space which is not large, DSN1COPY may produce unpredictable results.

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. 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 LARGE partitioned table space, even if the LARGE keyword is not specified.

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.

integer can range from 1 to 254.

PRINT(*hexadecimal-constant,hexadecimal-constant***)**
Causes the SYSUT1 data set to be printed in hexadecimal format on the SYSPRINT data set. You can enter 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 digits in length.

The following example shows how you code the PRINT parameter if you wanted 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, they apply to the entire input file whether or not a range of pages is being printed.

PIECESIZ(*integer***)**
Specifies the maximum piece size (data set size) for non-partitioned indexes.

The defaults for PIECESIZ are 2G (2 GB) for indexes backed by non-LARGE table spaces and 4G (4 GB) for indexes backed by LARGE table spaces. 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 LARGE table space, then the LARGE option is required.

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 one of 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 one of 1, 2, or 4.

Valid values for piece size are shown in Table 68 on page 3-61.

Table 68. Valid Options for Piece Size

KB	MB	GB
	1	1
	2	2
	4	4[1]
	8	
	16	
	32	
	64	
	128	
256	256	
512	512	

Notes::

[1]Valid only for LARGE tablespaces

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 printed in SYSPRINT. The VALUE parameter can be specified 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. If hexadecimal characters are specified, they must be enclosed in single quotation marks.

If you want to search your input file for the string '12345', your JCL might look like this:

```
//STEP1 EXEC PGM=DSN1COPY,PARM='VALUE(12345)'
```

If you want to search for the equivalent hexadecimal character string, your JCL might 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 SYSXLAT file by using the DD cards. DSN1COPY can only translate up to 500 record OBIDs at a time. If you specify this option, CHECK processing is performed regardless of whether you specify the CHECK option.

RESET

Causes the log RBAs in each index 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 only 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 zero 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 *Administration Guide*.

Chapter 3-8. DSN1LOGP

The DSN1LOGP utility formats the contents of the recovery log for display. There are two report formats:

- 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 Diagram: For a diagram of DSN1LOGP syntax and a description of available options, see “DSN1LOGP Syntax” on page 3-77.

Before Running DSN1LOGP

This section describes considerations you need to know before you run DSN1LOGP.

Environment

DSN1LOGP runs as a batch MVS job.

DSN1LOGP can be used 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 are RACF-protected, the authorization ID of the job must have RACF authority. Similarly, if the necessary data sets are password protected, the authorization ID of the job must have the appropriate VSAM passwords.

Reading Archive Log Data Sets on Tape

If your archive logs are stored on tape, during the archiving process two files are constructed on tape. The first file is the BSDS, and the second is a dump of the active log 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 MVS JES statements shown in Table 69 and Table 70.

Table 69. JES2 Environment JCL for Tape Archive Log Data Sets

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	Placed on the JOB card, it performs the same function as /*HOLD

Table 70. JES3 Environment JCL for Tape Archive Log Data Sets

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	Placed on the JOB card, it performs the same function as //*MAIN HOLD=YES

Alternatively, you can submit the job to an MVS initiator which has been established by your operations center for exclusive use by jobs that require tape mounts. The initiator class can be specified using the CLASS parameter on the JOB card, in both a JES2 and a JES3 environment.

For additional information on these options, refer to the *MVS/ESA JCL User's Guide* or the *MVS/ESA JCL Reference.*

Locating Table and Index Identifiers

DSN1PRNT can be used to find the DBIDs, PSIDs, ISOBIDs, and OBIDs of the tables and indexes from the system tables.

Creating the JCL to Run DSN1LOGP

See “DSN1LOGP Syntax” on page 3-77 for DSN1LOGP syntax and option descriptions.

Required Data Sets

When you invoke 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 3-78. 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. LRECL is 131. For an example of the appropriate JCL, see page 3-68.

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 *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 page 3-66 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 indicate 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 BSDS). DSN1LOGP scans all active log data sets the job indicates, as long as they are in the correct log RBA sequence.

See page 3-66 for guidance in using these DD statements.

ARCHIVE If the BSDS is not available (as described under ACTIVE*n*, above), you can indicate which archive log data sets are to be processed by specifying one ARCHIVE DD statement, concatenated with one or more DD statements as shown on page 3-67.

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, as long as they are in the correct log RBA sequence.

See page 3-67 for guidance in using the ARCHIVE DD statement.

Data Sharing Requirements: When selecting log records from more than one DB2 subsystem, you must use 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 *Administration Guide* for descriptions of those statements. If you use GROUP or M*xx*BSDSs to locate the log data sets, then you must use LRSNSTART to define the selection range.

Sample JCL and 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 it 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 3-27 and in Section 4 (Volume 1) of *Administration Guide*. During normal DB2 operation, messages are issued that include information about log RBAs.

Example 2: Using DSN1LOGP on 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, each log data set must be run 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 have been made to all table spaces and index spaces in the database whose DBID is X'4' by the specified unit of recovery.

```
//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 in order 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 invoke 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 upon 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)
/*
```

Data Sharing Examples: 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)
/*
```

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)
/*
```

After Running DSN1LOGP

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 30 on page 3-70 shows a sample of the summary report. Figure 31 on page 3-72 shows a sample of the detail report. Figure 32 on page 3-76 shows a sample of data propagation information from a summary report. After each sample is a description of the output.

```
DSN1212I DSN1LGRD FIRST LOG LRSN ENCOUNTERED AA526968220D

================================================================================
DSN1150I SUMMARY OF COMPLETED EVENTS

DSN1151I DSN1LPRT  MEMBER=V51B     UR  CONNID=V51B      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=V51B     UR  CONNID=V51B      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=V51B
               STARTRBA=000000068CD3  ENDRBA=00000006CAED  STARTLRSN=AA527AA809DF  ENDLRSN=AA527AA829F4
               DATE=94.347  TIME=12:32:29

DSN1162I DSN1LPRT  MEMBER=V51C     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=V51A     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 30. Sample DSN1LOGP Summary Report

Description of Summary Report: The summary report 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: URs by the order of their completion and checkpoints 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, or IN-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 it 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 page sets modified is incomplete for that UR.

```
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

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
```

Figure 31 (Part 1 of 5). Sample DSN1LOGP Detail Report

```
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

0000033592A0  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

000000001138E  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
```

Figure 31 (Part 2 of 5). Sample DSN1LOGP Detail Report

```
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

00000003956D  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

 .....

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
```

Figure 31 (Part 3 of 5). Sample DSN1LOGP Detail Report

```
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

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
```

Figure 31 (Part 4 of 5). Sample DSN1LOGP Detail Report

```
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 31 (Part 5 of 5). Sample DSN1LOGP Detail Report

The *detail report* 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 sets log records, open and close data sets log records, set write, reset write, and page set write log records.
- *UR control log records for the complete or incomplete unit of recovery*.

The volume of the detail log records can be reduced by specifying an optional parameter.

Data Propagation Information in the Summary Report: Figure 32 shows a sample of information from the DSN1LOGP summary report about log records of changes to DB2 tables that were created with DATA CAPTURE CHANGES.

```
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 32. Sample Data Propagation Information from the Summary Report

The fields show the following:

- START RBA and END RBA show the first and last RBA 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 that there were not more than 10.
- LR WRITTEN shows the number of log records written that were 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. A value of 2147483647 means that an overflow occurred, and the count is not valid.

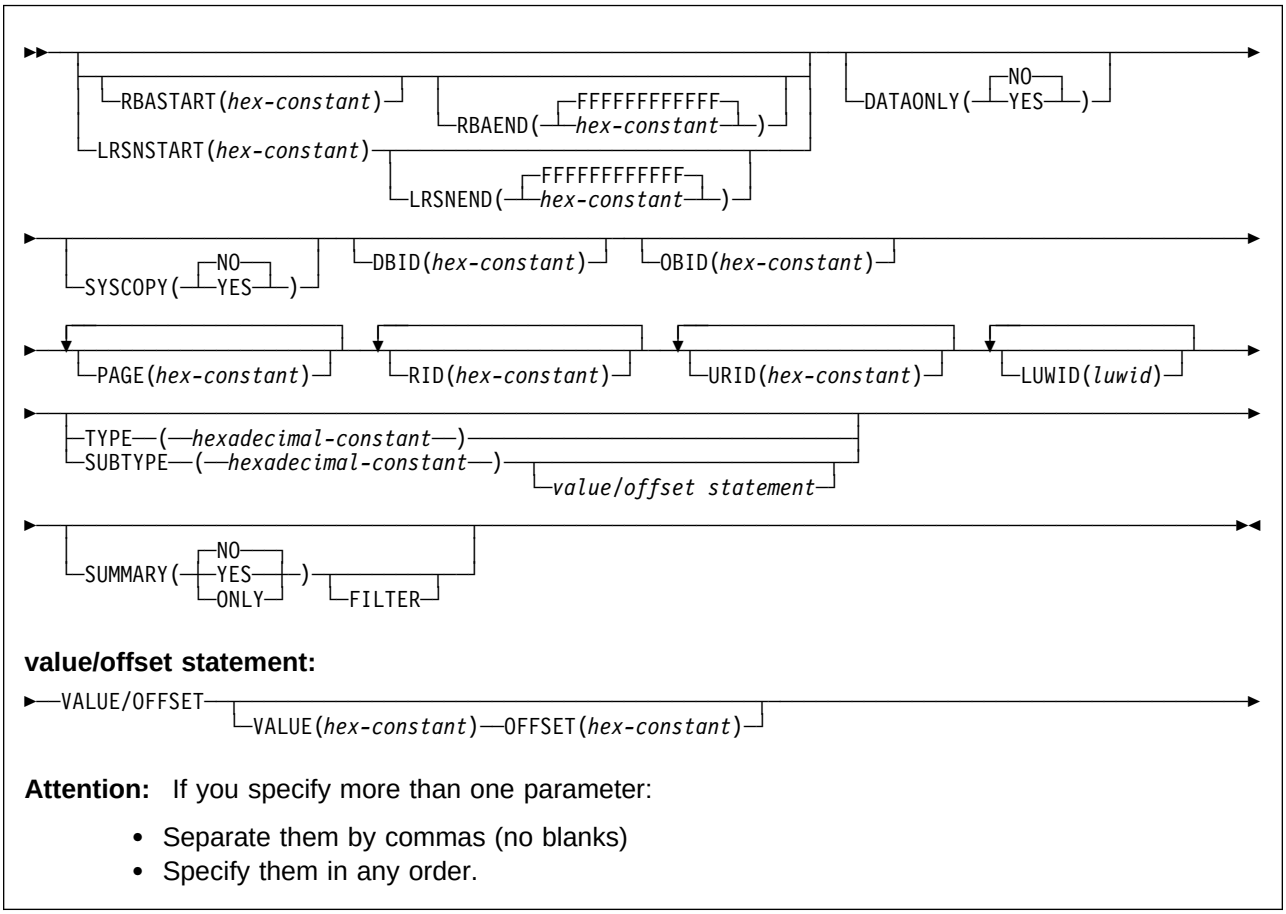
- LR RETRIEVED is the number of captured RBAs that were retrieved by DB2CDCEX. A value of 2147483647 means an overflow occurred, and the count is not valid.
- LR NOT RETRIEVED is the difference between the number of log records written (LR WRITTEN) and the number retrieved (LR RETRIEVED).

Receiving Error Codes and ABENDs

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 *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).

DSN1LOGP Syntax



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 JCL and Control Statements” on page 3-66.

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. Specify keywords in any order, except when using the keywords VALUE, OFFSET, and FILTER. The keyword SUBTYPE must be specified before VALUE and OFFSET. The keyword SUMMARY must be specified before FILTER.

RBASTART(*hexadecimal-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.

hexadecimal-constant can consist of 1 to 12 digits (6 bytes), and leading zeros are not required.

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

RBAEND(*hexadecimal-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.

hexadecimal-constant can consist of 1 to 12 digits (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 of 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 use the member BSDSs to locate the log data sets from more than one DB2 subsystem. You can specify either LRSNSTART or RBASTART to use the BSDS of a single DB2 subsystem to locate the log data sets.

LRSNEND(*hex-constant***)**
Specifies the LRSN value of the last log record to be scanned. The default when LRSNSTART is specified 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).

(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.

The **default** is **DATAONLY**(**NO**).

SYSCOPY
Limits the detail report to SYSCOPY log records.

(YES)
Includes only SYSCOPY log records in the detail report.

(NO)
Does not limit records to SYSCOPY records only.

The **default** is **SYSCOPY**(**NO**).

DBID(*hexadecimal-constant***)**
Specifies a hexadecimal database identifier (DBID). DSN1LOGP extracts only records associated with that DBID. For any given job, specify this keyword only once.

hexadecimal-constant can consist of 1 to 4 digits, and leading zeros are not required.

The DBID is displayed in many DB2 messages. You can also find the DBID in the DB2 catalog for a specific object (for example, in the column named “DBID” of the SYSIBM.SYSTABLESPACE catalog table).

When you select a DBID from a catalog table, the value is displayed in decimal format. Use the SQL HEX function in your select statement to convert them to hexadecimal:

```
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(*hexadecimal-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.

hexadecimal constant can consist of 1 to 4 digits, and 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 identifier of a file descriptor, and must not be confused with the PSID, which is the information you must include when you invoke 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, and must not be confused with the ISOBID, which is the information you must include when you invoke 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(*hexadecimal-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 data or index page changed. 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.

hexadecimal-constant can consist of a maximum of 8 digits.

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(*hexadecimal-constant***)**
Specifies a record identifier, which is a 10-digit hexadecimal number, with the first 8 digits representing the page number and the last 2 digits 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 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 records specified.

The PAGE and RID keywords cannot both be specified.

URID(*hexadecimal-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.

hexadecimal constant can consist of 1 to 12 digits (6 bytes), and 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 1- to 8-character network ID, a period, and a 1- to 8-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(*hexadecimal-constant***)**
Limits the log records extracted to records of a specified type. The TYPE and SUBTYPE options are mutually exclusive.

hexadecimal-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(*hexadecimal-constant***)**
Restricts formatting to a particular subtype of unit of recovery undo and redo log records 9types 200 and 400). The TYPE and SUBTYPE options are mutually exclusive.

hexadecimal-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	Type 2 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(*hexadecimal-constant***)**
Specifies a value that must appear in a log record to be extracted.

hexadecimal-constant can consist of a maximum of 64 characters and must be an even number of characters.

The SUBTYPE option must be specified before the VALUE option.

OFFSET(*hexadecimal-constant***)**
Specifies an offset from the log record header at which the value specified in the VALUE option must appear.

hexadecimal-constant can consist of a maximum of 8 characters.

The SUBTYPE option must be specified before specifying the OFFSET option.

SUMMARY
The report summarizes all recovery information within the RBASTART and RBAEND specifications. Summary information can be used 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.

(YES)
Generates both a detail and summary report.

(NO)
Generates only a detail report.

The **default** is **SUMMARY**(**NO**).

(ONLY)
Generates only a summary report.

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.

FILTER
Restricts the summary report to include messages for only those URIDs and LUWIDs specified. Specify this option only once.

The SUMMARY option must be specified before FILTER.

Chapter 3-9. DSN1PRNT

DSN1PRNT allows you to print DB2 VSAM data sets that contain table spaces or index spaces (including dictionary pages for compressed data). You can also print:

- 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 will be 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.

Syntax Diagram: For a diagram of DSN1PRNT syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 3-87.

Before Running DSN1PRNT

DSN1PRNT rather than DSN1COPY should be used for printing. This is because DSN1COPY will scan the whole SYSUT1 data set, but DSN1PRNT may be able to stop scanning before the end. Also, DSN1PRNT can write a formatted dump.

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, you should be sure 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 are RACF-protected, the authorization ID of the job must have RACF authority. Also, if the data sets are password protected, the authorization ID of the job must have the appropriate VSAM passwords to execute this utility.

Creating the JCL to Run DSN1PRNT

See “Syntax and Options of the Control Statement” on page 3-87 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 or index space that is larger than 2 gigabytes, or is a partitioned table 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 4 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).

Sample JCL and Control Statements

Sample JCL for 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
```

Sample JCL 2: Print a Non-partitioned Index with 64M piece size. Index:

```
//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
```

After Running DSN1PRNT

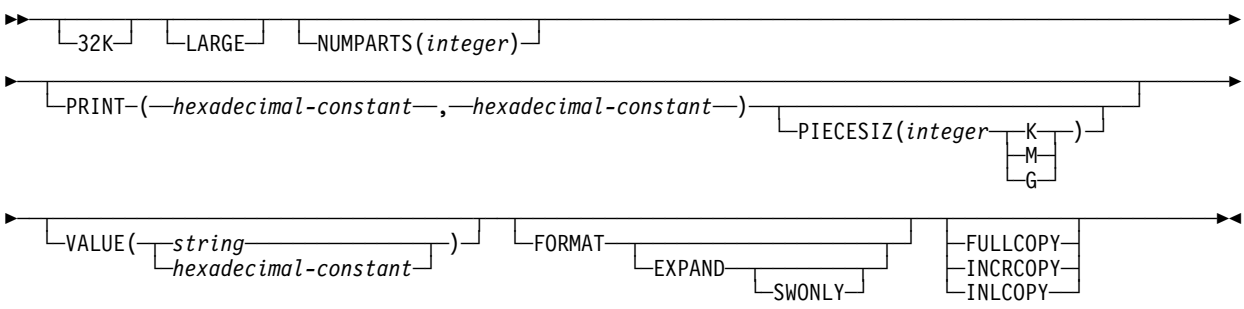
Interpreting Output

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.

When requested to print a page or 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 correct.

Syntax and Options of the Control Statement

DSN1PRNT Syntax



Attention: 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 4KB.

LARGE
Specifies that the input data set has been defined as a large partitioned table space or an index data set backed by a large partitioned table space. If the LARGE option is omitted for a LARGE partitioned table space or specified for a table space which is not LARGE, DSN1PRNT may produce unpredictable results.

NUMPARTS(*integer*)
Specifies the number of a partition 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 LARGE partitioned table space, even if the LARGE keyword is not specified.

DSN1PRNT cannot always validate the NUMPARTS parameter. If you specify it incorrectly, DSN1PRNT can 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 enter 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 digits in length.

The following example shows how to code the PRINT parameter if you wanted 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 2-253.

PIECESIZ(*integer***)**
Specifies the maximum piece size (data set size) for non-partitioned indexes.

The defaults for PIECESIZ are 2G (2 GB) for indexes backed by non-LARGE table spaces and 4G (4 GB) for indexes backed by LARGE table spaces. 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 LARGE table space, then the LARGE option is required.

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 one of 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 one of 1, 2, or 4.

Valid values for piece size are shown in Table 71 on page 3-89.

Table 71. Valid Options for Piece Size

KB	MB	GB
	1	1
	2	2
	4	4[1]
	8	
	16	
	32	
	64	
	128	
256	256	
512	512	

Notes::

[1]Valid only for LARGE tablespaces

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. The VALUE parameter can be specified 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. If hexadecimal characters are specified, they must be enclosed in single quotation marks.

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 in order 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 miscellaneous 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 miscellaneous error messages or unpredictable results.

INLCOPY
Specifies that the input data is an inline copy data set.

Chapter 3-10. DSN1SDMP

Under the direction of IBM service, the IFC Selective Dump (DSN1SDMP) utility allows you 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 *Administration Guide*.

Syntax Diagram: For a diagram of DSN1SDMP syntax and a description of available options, see “Syntax and Options of the Control Statement” on page 3-95.

Before Running DSN1SDMP

Environment

You run DSN1SDMP as an MVS job and invoke it with the DSN TSO command processor. To invoke 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:

- TRACE system privilege
- SYSOPR authority
- SYSADM authority
- MONITOR1 or MONITOR2 privileges, if you are using user-defined data sets.

The user who invokes DSN1SDMP must have EXECUTE authority on the plan specified in the *trace-parameters* of the START TRACE keyword.

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. That avoids 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 3-96). 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.

The buffer size can be specified 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 trace records generated requires a larger buffer size than was specified, you can lose some trace records. If this happens, you will receive error message DSN2724I.

Causing a Dump

All of the following must occur before DSN1SDMP causes a DB2 dump:

- DB2 has produced a trace record that satisfies all of the selection criteria
- An abend action (ABENDRET or ABENDTER) has been specified
- The AFTER and FOR conditions for the trace are satisfied.

If these three things occur, then an `00E601xx` abend occurs. *xx* is an integer between 1 and 99, obtained from the user-specified value on the ACTION keyword.

Creating the JCL to Run DSN1SDMP

See “Syntax and Options of the Control Statement” on page 3-95 for DSN1SDMP syntax and option descriptions.

Required Data Sets

DD Cards: DSN1SDMP uses the DD cards described below:

SDMPIN DD — Defines the control data set which 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 DD — 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 DD — Defines the data set to contain an ABEND dump in case DSN1SDMP abends. This DD card is optional.

SDMPTRAC DD — 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 DD — Defines the DSN commands to connect to DB2 and to invoke 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, it 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 invoked without a PLAN, DSN generates an error if no DSN1SDMP plan exists for which the user has execute authority.

Sample JCL and Control Statements

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))
/*
```

Stopping DSN1SDMP

If you must stop DSN1SDMP, use the STOP TRACE command.

If DSN1SDMP does not finish execution, it can be stopped by a operator STOP TRACE command, for example:

```
-STOP TRACE=P CLASS(32)
```

A STOP TRACE or MODIFY TRACE command entered from an MVS console against the trace started by DSN1SDMP causes immediate abnormal termination of DSN1SDMP processing, with these effects:

- The IFI READA function issued by DSN1SDMP terminates
- The STOP TRACE command issued by DSN1SDMP fails if, before the DSN1SDMP command is issued, either the original trace is terminated or modify trace processing completes

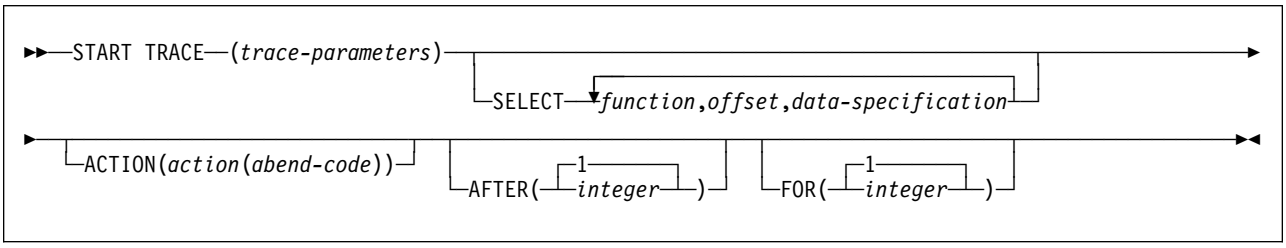
After Running DSN1SDMP

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.

Syntax and Options of the Control Statement

DSN1SDMP Syntax



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 *Command Reference*, except you do not use the subsystem recognition character.

If the START TRACE command in the SDMPIN input stream is illegal, 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, then 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. The remaining selective dump keywords can be specified in any order following START TRACE.

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 8 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 purpose, 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 *Administration Guide*.

The selection criteria are specified through 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 offset specified. 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 offset specified. The offset is always calculated from the current record pointer. The test will succeed if the data from the offset specified is greater than or equal to *data-specification* 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 offset specified. The offset is always calculated from the current record pointer. The test will succeed if the data from the offset specified is less than or equal to *data-specification* 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 pointer that many 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 located in the first four bytes of the record. P4,00 reads that offset and moves the current pointer to the start of the standard header.

- LN - Advances the current pointer by the number of bytes indicated in the 2 byte field located *offset* bytes from the previous current 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 an GE, LE, LN or DR.

The format of the DB2 trace record at *data-specification* comparison time is:

Figure 33. 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 section depend on the IFCID being recorded. There can be one or more data sections per record. 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 *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 ABEND 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**).

Appendixes

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 72. Identifier Length Limits

Item	Limit
Longest alias, synonym, collection ID, correlation name, statement name, or name of a column, cursor, index, table, view, or table check constraint	18 bytes
Longest authorization name, package name, or name of a plan, database, table space, storage group, or referential constraint	8 bytes
Longest host identifier	64 bytes
Longest server name or location identifier	16 bytes

Table 73. Numeric Limits

Item	Limit
Smallest INTEGER value	-2147483648
Largest INTEGER value	2147483647
Smallest SMALLINT value	-32768
Largest SMALLINT value	32767
Largest decimal precision	31
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	$10^{31} - 1$

Table 74 (Page 1 of 2). String Length Limits

Item	Limit
# Maximum length of CHAR	255 bytes
Maximum length of GRAPHIC	127 characters
Maximum length of VARCHAR[4]	4046 bytes, for 4KB pages 32704 bytes, for 32KB pages
Maximum length of VARGRAPHIC[4]	4046 bytes (2023 DBCS characters), for 4KB pages 32704 bytes (16352 DBCS characters), for 32KB pages
# Maximum length of a character constant	255 bytes

[4] 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 74 (Page 2 of 2). String Length Limits

Item	Limit
Maximum length of a hexadecimal constant	254 digits
Maximum length of a graphic string constant	124 characters
Maximum length of a concatenated character string	32764 bytes
Maximum length of a concatenated graphic string	16382 DBCS characters

Table 75. 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 76 (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)	750 or fewer 749 if the table is a dependent
Maximum number of base tables in a view	15
Maximum row and record sizes for a table	See the description of CREATE TABLE in Chapter 6 of *SQL Reference*
Maximum number of volumes in an SMS-managed or DB2 user-defined data set	Limited by MVS to 59
Maximum number of partitions in a partitioned table space or partitioned index	64 for table spaces that are not large 254 for table spaces that are large
Maximum size of a partition (table space or index)	For table spaces that are not large: 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 large: 4 gigabytes, for 1 to 254 partitions
Maximum size of a DBRM entry	131072 bytes
Longest index key	254 bytes less the number of key columns that allow nulls. See the description of CREATE INDEX in Chapter 6 of *SQL Reference*.
Maximum number of bytes used in the partitioning of a partitioned index[5]	40

[5] If the key of a partitioned index is longer than 40 bytes, only the first 40 bytes are used to determine the high value for each partition.

Table 76 (Page 2 of 2). DB2 Limits on SQL Statements

Item	Limit
Maximum number of columns in an index key	64
Maximum number of tables and views that can be identified in a subselect	15 or fewer, depending on the complexity of the subselect
Maximum total length of host and indicator variables pointed to in an SQLDA	32767 bytes
Longest host variable used for insert or update	32704 bytes
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 parameters of a stored procedure and any CALL statement referencing the procedure	As many as can be defined by the parameter list stored in SYSPROCEDURES.PARMLIST. The maximum length of the column is 3000 bytes.
Maximum number of bytes that can be passed in a single parameter of an SQL CALL statement	32765 bytes

Table 77 (Page 1 of 2). 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	1016 gigabytes
Largest log space	2^{48}
Largest active log data set	2 gigabtyes
Largest archive log data set	2 gigabtyes
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	32511
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

Table 77 (Page 2 of 2). DB2 System Limits

Item	Limit
Maximum partitioned data set size	See item “maximum size of a partition” in Table 76 on page X-4

Stored procedures shipped with DB2

DB2 provides several stored procedures that you can call in your application programs to perform a number of utility functions. Those stored procedures are:

- The utilities stored procedure (DSNUTILS)

  This stored procedure lets you invoke utilities from a local or remote client program. See “Invoking utilities as a stored procedure (DSNUTILS)” for information.

- The DB2 UDB Control Center table space and index information stored procedure (DSNACCQC)

  This stored procedure helps you determine when utilities should be run on your databases. This stored procedure is designed primarily for use by the DB2 UDB Control Center but can be invoked from any client program. See “The DB2 UDB Control Center table space and index information stored procedure (DSNACCQC)” on page X-17 for information.

- The DB2 UDB Control Center partition information stored procedure (DSNACCAV)

  This stored procedure helps you determine when utilities should be run on your partitioned table spaces. This stored procedure is designed primarily for use by the DB2 UDB Control Center but can be invoked from any client program. See “The DB2 UDB Control Center partition information stored procedure (DSNACCAV)” on page X-25 for information.

Invoking utilities as a stored procedure (DSNUTILS)

The DSNUTILS stored procedure lets 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 global temporary table (SYSIBM.SYSPRINT)
- Captures the utility output stream (SYSPRINT) into a global 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.

The JCL for installing the DSNUTILS stored procedure is provided in installation job DSNTIJSG.

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.

If the DSNUTILS stored procedure invokes a new utility, refer to Table 78 on page X-9 for information about the default data dispositions specified for dynamically-allocated data sets.

Table 78. Data dispositions for dynamically allocated datasets

ddname	CHECK DATA	CHECK INDEX	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	NEW DELETE CATLG	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

If the DSNUTILS stored procedure restarts a current utility, refer to Table 79 on page X-10 for information about the default data dispositions specified for dynamically-allocated data sets.

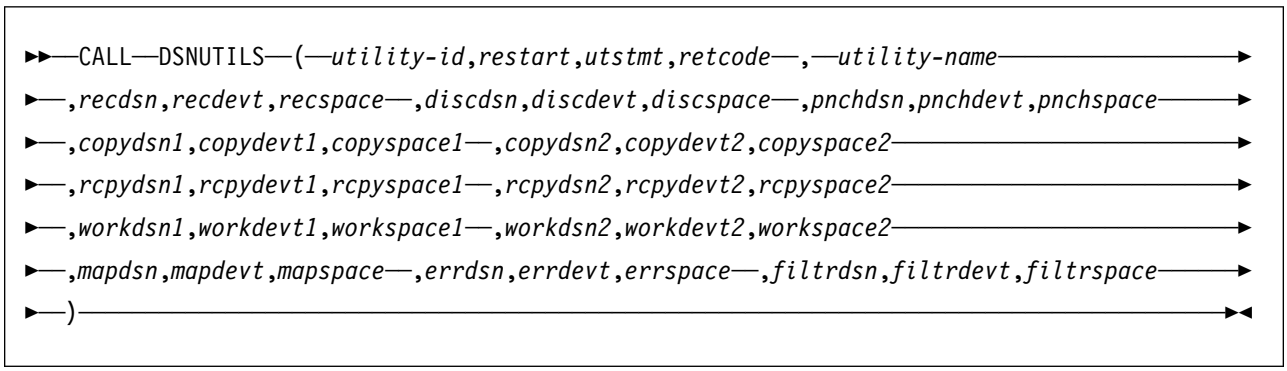
Table 79. Data dispositions for dynamically allocated datasets on RESTART

ddname	CHECK DATA	CHECK INDEX	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	MOD DELETE CATLG	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

DSNUTILS syntax diagram

For guidance in interpreting syntax diagrams, see “How to Read the Syntax Diagrams” on page 1-3.

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
- COPY
- DIAGNOSE
- LOAD
- MERGECOPY
- MODIFY RECOVERY
- QUIESCE
- REBUILD INDEX
- RECOVER
- REORG INDEX
- 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 X-7.

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.

The DB2 UDB Control Center table space and index information stored procedure (DSNACCQC)

The information under this heading is Product-sensitive Programming Interface and Associated Guidance Information, as defined in “Notices” on page ix.

The DSNACCQC stored procedure gives you information about your table spaces and indexes. You can use DSNACCQC to obtain the following types of information:

- Table spaces and indexes on which RUNSTATS needs to be run
- Table spaces and indexes on which the STOSPACE utility has not been run
- Table spaces and indexes that exceed the primary space allocation
- Table spaces with more than a user-specified percentage of relocated rows
- Table spaces with more than a user-specified percentage of space that is occupied by dropped tables
- Table spaces with table space locking
- Simple table spaces with more than one table
- Indexes with clustering problems
- Indexes with more than a user-specified number of index levels
- Indexes with more than a user-specified LEAFDIST value
- Type 1 indexes
- Nonunique indexes with long RID chains
- Indexes that are not used in static SQL statements

Environment

DSNACCQC runs in a DB2-established stored procedures address space.

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 DSNACCQC
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

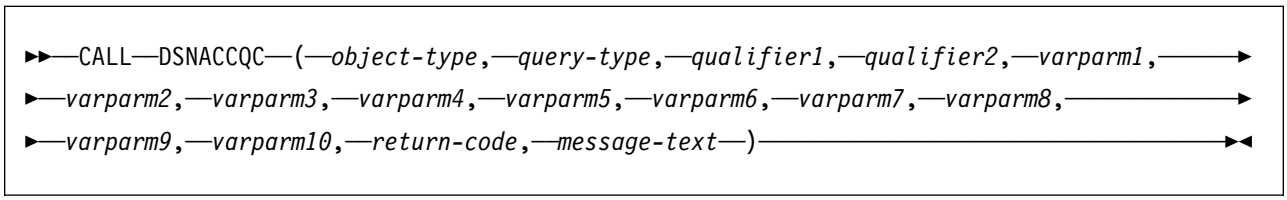
The owner of the package or plan that contains the CALL statement must also have SELECT authority on the following catalog table spaces:

- SYSIBM.SYSINDEXES
- SYSIBM.SYSINDEXPART
- SYSIBM.SYSPACKDEP
- SYSIBM.SYSPLANDEP

- SYSIBM.SYSTABLEPART
- SYSIBM.SYSTABLES
- SYSIBM.SYSTABLESPACE

DSNACCQC syntax diagram

The following syntax diagram shows the SQL CALL statement for invoking DSNACCQC. Because the linkage convention for DSNACCQC is SIMPLE, you cannot pass null values for input parameters. For character parameters that you are not using, specify an empty string (''').



DSNACCQC option descriptions

object-type
Specifies whether you want information about indexes or table spaces. This is an input parameter of type INTEGER. The contents must be one of the following values:

0 Obtain information about indexes

1 Obtain information about table spaces

query-type
Specifies the type of information that you want to obtain. This is an input parameter of type INTEGER.

If *object-type* is 0 (indexes), *query-type* must be one of the following values:

0 Obtains information about indexes on which RUNSTATS needs to be run.

1 Obtains information about indexes with clustering problems.

2 Obtains information about indexes with more than a user-specified number of levels.

3 Obtains information about indexes with a LEAFDIST value that is higher than a user-specified percentage.

4 Obtains information about type 1 indexes.

5 Obtains information about nonunique indexes with long RID chains.

6 Obtains information about indexes that are not used in static SQL statements.

7 Obtains information about indexes on which the STOSPACE utility has not been run.

8 Obtains information about indexes that have exceeded their allocated primary space quantity.

If *object-type* is 1 (table spaces), *query-type* must be one of the following values:

0 Obtains information about table spaces on which RUNSTATS needs to be run.

1 Obtains information about table spaces with more than a user-specified percentage of relocated rows.

2 Obtains information about table spaces with more than a user-specified percentage of space that is occupied by dropped tables.

3 Obtains information about table spaces with table space locking.

4 Obtains information about simple table spaces with more than one table.

5 Obtains information about table spaces on which the STOSPACE utility has not been run.

6 Obtains information about table spaces that have exceeded their allocated primary space quantity.

qualifier1
Narrows the search for objects that match *query-type* to a specified set of database names. This is an input parameter of type VARCHAR(255).

The format of this parameter is the same as the format of *pattern-expression* in an SQL LIKE predicate. *pattern-expression* is described in Chapter 5 of *SQL Reference*.

For example, to obtain information about table spaces or indexes in all databases with names that begin with ACCOUNT, specify this value for *qualifier1:*

```
ACCOUNT%
```

qualifier2
Narrows the search for objects that match *query-type* to a specified set of creator names. A creator name is the value of column CREATOR in SYSIBM.SYSTABLESPACE for table space queries, or SYSIBM.SYSINDEXES for index queries. This is an input parameter of type VARCHAR(255).

The format of this parameter is the same as the format of *pattern-expression* in an SQL LIKE predicate. *pattern-expression* is described in Chapter 5 of *SQL Reference*.

For example, to obtain information about table spaces or indexes with creators that begin with DSN8, specify this value for *qualifier2:*

```
DSN8%
```

varparm1, *varparm2*, *varparm3*
The meanings of these parameters vary with *object-type* and *query-type*. See Table 80 on page X-20 for the meaning of of each parameter for table space queries. See Table 81 on page X-21 for the meaning of of each parameter for index queries.

These are input parameters of type VARCHAR(255).

varparm4 **through** *varparm10*
These variables are reserved for future use. Specify an empty string (' ') for each parameter value.

These are input parameters of type VARCHAR(255).

return-code
Specifies the return code from the DSNACCQC call, which is one of the following values:

0 DSNACCQC executed successfully.

12 An error occurred during DSNACCQC execution.

return-code is an output parameter of type INTEGER.

message-text
If an error occurs while DSNACCQC executes, contains information about the error. If the error is an SQL error, *message-text* also contains the formatted SQLCA. The message text consists of from one to eleven lines, each with a length of 121 bytes. the last byte of each line is a new-line character. *message-text* is an output parameter of type VARCHAR(1331).

Table 80 (Page 1 of 2). Variable input parameter values for DSNACCQC table space queries

query-type	**Parameter**	**Value**
0	*varparm1*	Timestamp in character format (*yyyy-mm-dd-hh.mm.ss.nnnnnn*). DSNACCQC returns information about table spaces on which RUNSTATS was run before this time or was never run.
	varparm2	Not used. Specify an empty string (' ').
	varparm3	Not used. Specify an empty string (' ').
1	*varparm1*	Character representation of a number between 0 and 100, which indicates the maximum acceptable percentage of relocated table rows. DSNACCQC returns information about table spaces for which (((FARINDREF+NEARINDREF)/CARD)*100)>*varparm1*.
	varparm2	Not used. Specify an empty string (' ').
	varparm3	Not used. Specify an empty string (' ').
2	*varparm1*	Character representation of a number between 0 and 100, which indicates the maximum acceptable percentage space that is occupied by rows of dropped tables. DSNACCQC returns information about table spaces for which PERCDROP>*varparm1*.
	varparm2	Not used. Specify an empty string (' ').
	varparm3	Not used. Specify an empty string (' ').
3	*varparm1*	Not used. Specify an empty string (' ').
	varparm2	Not used. Specify an empty string (' ').
	varparm3	Not used. Specify an empty string (' ').
4	*varparm1*	Not used. Specify an empty string (' ').
	varparm2	Not used. Specify an empty string (' ').
	varparm3	Not used. Specify an empty string (' ').
5	*varparm1*	Not used. Specify an empty string (' ').
	varparm2	Not used. Specify an empty string (' ').
	varparm3	Not used. Specify an empty string (' ').

Table 80 (Page 2 of 2). Variable input parameter values for DSNACCQC table space queries

query-type	**Param-eter**	**Value**
6	*varparm1*	Not used. Specify an empty string (' ').
	varparm2	Not used. Specify an empty string (' ').
	varparm3	Not used. Specify an empty string (' ').

Table 81 (Page 1 of 2). Variable input parameter values for DSNACCQC index queries

query-type	**Param-eter**	**Value**
0	*varparm1*	Timestamp in character format (*yyyy-mm-dd-hh.mm.ss.nnnnnn*). DSNACCQC returns information about indexes on which RUNSTATS was run before this time or was never run.
	varparm2	Not used. Specify an empty string (' ').
	varparm3	Not used. Specify an empty string (' ').
1	*varparm1*	Character representation of a number between 0 and 100, which indicates the maximum acceptable percentage of table rows that are from from their optimal position. DSNACCQC returns information about indexes for which ((FAROFFPOSF/CARDF)*100)>*varparm1*.
	varparm2	Character representation of a number between 0 and 100, which indicates the maximum acceptable percentage of table rows that are near but not at their optimal position. DSNACCQC returns information about indexes for which ((NEAROFFPOSF/CARDF)*100)>*varparm2*.
	varparm3	Character representation of a number between 0 and 100, which indicates the minimum acceptable percentage of table rows that are in clustering order. DSNACCQC returns information about indexes for which CLUSTERRATIO<*varparm3*.
2	*varparm1*	Character representation of a number that indicates the maximum acceptable number of index levels. DSNACCQC returns information about indexes for which NLEVELS>*varparm1*.
	varparm2	Not used. Specify an empty string (' ').
	varparm3	Not used. Specify an empty string (' ').
3	*varparm1*	Character representation of a number that indicates the maximum acceptable value for 100 times the average number of leaf pages between successive active leaf pages of the index. DSNACCQC returns information about indexes for which LEAFDIST>*varparm1*.
	varparm2	Not used. Specify an empty string (' ').
	varparm3	Not used. Specify an empty string (' ').
4	*varparm1*	Not used. Specify an empty string (' ').
	varparm2	Not used. Specify an empty string (' ').
	varparm3	Not used. Specify an empty string (' ').

Table 81 (Page 2 of 2). Variable input parameter values for DSNACCQC index queries

query-type	**Param-eter**	**Value**
5	*varparm1*	Character representation of a number that indicates the maximum acceptable average length for RID chains. DSNACCQC returns information about indexes for which ((CARDF*1.0)/FULLKEYCARDF)>*varparm1*.
	varparm2	Not used. Specify an empty string ('').
	varparm3	Not used. Specify an empty string ('').
6	*varparm1*	Not used. Specify an empty string ('').
	varparm2	Not used. Specify an empty string ('').
	varparm3	Not used. Specify an empty string ('').
7	*varparm1*	Not used. Specify an empty string ('').
	varparm2	Not used. Specify an empty string ('').
	varparm3	Not used. Specify an empty string ('').
8	*varparm1*	Not used. Specify an empty string ('').
	varparm2	Not used. Specify an empty string ('').
	varparm3	Not used. Specify an empty string ('').

Example of DSNACCQC invocation

Suppose that you want information on indexes on which RUNSTATS has never been run. You want information only on indexes in databases whose names begin with DSNCC. The parameter declarations and DSNACCQC call looks like this:

```
#                   DCL OBJTYPE    FIXED BIN(31);
#                   DCL QUERY      FIXED BIN(31);
#                   DCL DBQUAL     CHAR(255) VARYING;
#                   DCL CREATQUAL  CHAR(255) VARYING;
#                   DCL VARPARM1   CHAR(255) VARYING;
#                   DCL VARPARM2   CHAR(255) VARYING;
#                   DCL VARPARM3   CHAR(255) VARYING;
#                   DCL VARPARM4   CHAR(255) VARYING;
#                   DCL VARPARM5   CHAR(255) VARYING;
#                   DCL VARPARM6   CHAR(255) VARYING;
#                   DCL VARPARM7   CHAR(255) VARYING;
#                   DCL VARPARM8   CHAR(255) VARYING;
#                   DCL VARPARM9   CHAR(255) VARYING;
#                   DCL VARPARM10  CHAR(255) VARYING;
#                   DCL RC         FIXED BIN(31);
#                   DCL MSGTEXT    CHAR(1331) VARYING;
#                   DCL IXTABLE SQL TYPE IS RESULT_SET_LOCATOR VARYING;

#                   OBJTYPE=0;
#                   QUERY=0;
#                   DBQUAL='DSNCC%';
#                   CREATQUAL='%';
#                   VARPARM1='0001-01-01-00.00.00.000000';
#                   VARPARM2='';
#                   VARPARM3='';
#                   VARPARM4='';
#                   VARPARM5='';
#                   VARPARM6='';
#                   VARPARM7='';
#                   VARPARM8='';
#                   VARPARM9='';
#                   VARPARM10='';

#                   EXEC SQL CALL SYSPROC.DSNACCQC(:OBJTYPE, :QUERY, :DBQUAL,
#                    :CREATQUAL, :VARPARM1, :VARPARM2,  :VARPARM3,
#                    :VARPARM4,  :VARPARM5, :VARPARM6,  :VARPARM7,
#                    :VARPARM8,  :VARPARM9, :VARPARM10,
#                    :RC, :MSGTEXT);
```

DSNACCQC output

In addition to the output parameters described in “DSNACCQC option descriptions” on page X-18, DSNACCQC returns one result set. The format of the result set varies, depending on whether you are retrieving index information (*object-type*=0) or table space information (*object-type*=1). Table 82 on page X-24 shows the columns of a result set row and the DB2 catalog table that is the source of information for each column for table space queries. Table 83 on page X-25 shows the columns of a result set row and the DB2 catalog table that is the source of information for index queries.

Table 82. Result set columns for DSNACCQC table space queries

Column name	Data type	DB2 catalog table that is the data source
NAME	CHAR(8)	SYSTABLESPACE
CREATOR	CHAR(8)	SYSTABLESPACE
BPOOL	CHAR(8)	SYSTABLESPACE
LOCKRULE	CHAR(1)	SYSTABLESPACE
LOCKMAX	INTEGER	SYSTABLESPACE
CLOSERULE	CHAR(1)	SYSTABLESPACE
ENCODING_SCHEME	CHAR(1)	SYSTABLESPACE
LOCKPART	CHAR(1)	SYSTABLESPACE
MAXROWS	SMALLINT	SYSTABLESPACE
PARTITIONS	SMALLINT	SYSTABLESPACE
TYPE	CHAR(1)	SYSTABLESPACE
SEGSIZE	SMALLINT	SYSTABLESPACE
SPACE	INTEGER	SYSTABLESPACE
NTABLES	SMALLINT	SYSTABLESPACE
STATUS	CHAR(1)	SYSTABLESPACE
STATSTIME	TIMESTAMP	SYSTABLESPACE
ERASERULE	CHAR(1)	SYSTABLESPACE
DBNAME	CHAR(8)	SYSTABLESPACE
DSETPASS	CHAR(8)	SYSTABLESPACE
Reserved	CHAR(1)	
Reserved	INTEGER	

Table 83. Result set columns for DSNACCQC index queries

Column name	Data type	DB2 catalog table that is the data source
CREATOR	CHAR(8)	SYSINDEXES
NAME	VARCHAR(18)	SYSINDEXES
TBCREATOR	CHAR(8)	SYSINDEXES
TBNAME	VARCHAR(18)	SYSINDEXES
UNIQUERULE	CHAR(1)	SYSINDEXES
INDEXTYPE	CHAR(1)	SYSINDEXES
INDEXSPACE	CHAR(8)	SYSINDEXES
CLUSTERING	CHAR(1)	SYSINDEXES
ERASERULE	CHAR(1)	SYSINDEXES
CLOSERULE	CHAR(1)	SYSINDEXES
COLCOUNT	SMALLINT	SYSINDEXES
DBID	SMALLINT	SYSINDEXES
DBNAME	CHAR(8)	SYSINDEXES
BPOOL	CHAR(8)	SYSINDEXES
PGSIZE	SMALLINT	SYSINDEXES
DSETPASS	CHAR(8)	SYSINDEXES
PIECESIZE	SMALLINT	SYSINDEXES
Reserved	CHAR(1)	
PARTITION_COUNT	INTEGER	SYSINDEXPART[1]

Notes:

1. The value of PARTITION_COUNT is COUNT(DISTINCT PARTITION) for partitioning indexes or 0 for nonpartitioning indexes. The PARTITION column is in SYSIBM.SYSINDEXPART.

To obtain the information from the result set, you can write your client program to retrieve information from a one result set with known contents. However, for greater flexibility, you might want to write your client program to retrieve data from an unknown number of result sets with unknown contents. Both techniques are shown in Section 6 of *Application Programming and SQL Guide*.

The DB2 UDB Control Center partition information stored procedure (DSNACCAV)

The information under this heading is Product-sensitive Programming Interface and Associated Guidance Information, as defined in “Notices” on page ix.

The DSNACCAV stored procedure gives you information about partitions in your table spaces and indexes. You can use DSNACCAV to obtain the following types of information:

- Partitions that need to be image copied
- Partitions that are in a restricted state
- Partitions on which RUNSTATS needs to be run
- Partitions on which REORG needs to be run

- Partitions that exceed a user-specified number of extents

Environment

DSNACCAV runs in a DB2-established stored procedures address space.

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:

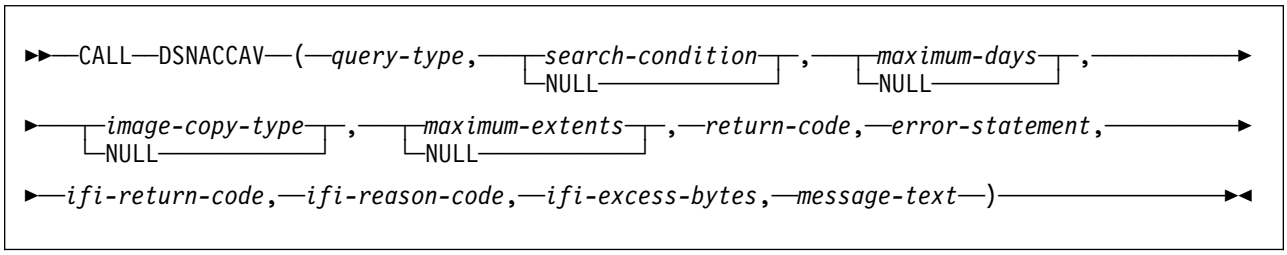
- The EXECUTE privilege on the package for DSNACCAV
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

The owner of the package or plan that contains the CALL statement must also have SELECT authority on the following catalog table spaces:

- SYSIBM.SYSCOPY
- SYSIBM.SYSINDEXES
- SYSIBM.SYSINDEXPART
- SYSIBM.SYSTABLEPART
- SYSIBM.SYSTABLES
- SYSIBM.SYSTABLESPACE

DSNACCAV syntax diagram

The following syntax diagram shows the SQL CALL statement for invoking DSNACCAV. Because the linkage convention for DSNACCAV is SIMPLE WITH NULLS, if you pass parameters in host variables, you need to include a null indicator with every host variable. Null indicators for input host variables must be initialized before you execute the CALL statement.



DSNACCAV option descriptions

query-type

Specifies the type of information that you want to obtain. This is an input parameter of type VARCHAR(20). The contents must be one of the following values:

COPY TABLESPACE

Obtains information about table space partitions for which image copies need to be made.

RESTRICT TABLESPACE
Obtains information about table space partitions that are in a restricted status.

RESTRICT INDEX
Obtains information about index partitions that are in a restricted status.

RUNSTATS TABLESPACE
Obtains information about table space partitions on which RUNSTATS needs to be run.

RUNSTATS INDEX
Obtains information about index partitions on which RUNSTATS needs to be run.

REORG TABLESPACE
Obtains information about table space partitions on which REORG needs to be run.

REORG INDEX
Obtains information about index partitions on which REORG needs to be run.

EXTENTS TABLESPACE
Obtains information about table space partitions that have used more than a user-specified number of extents.

EXTENTS INDEX
Obtains information about index partitions that have used more than a user-specified number of extents.

search-condition
Narrows the search for objects that match *query-type.* This is an input parameter of type VARCHAR(4096).

The format of this parameter is the same as the format of *search-condition* in an SQL *where-clause. search-condition* is described in Chapter 5 of *SQL Reference*.

If the call is executed to obtain table space information, *search-condition* can include any column in SYSIBM.SYSTABLESPACE. If the call is executed to obtain index information, *search-condition* can include any column in SYSIBM.SYSINDEXES. Each column name must be preceded by the string 'A.'.

For example, to obtain information about table spaces with creator ADMF001, specify this value for *search-condition:*

```
A.CREATOR='ADMF001'
```

maximum-days
Specifies the maximum number of days that should elapse between executions of the REORG, RUNSTATS, or COPY utility. DSNACCAV uses this value as the criterion for determining which table space or index partitions need to have the utility that you specified in *query-type* run against them. This value can be specified if *query-type* has one of the following values:

- COPY TABLESPACE
- RUNSTATS TABLESPACE
- RUNSTATS INDEX

- REORG TABLESPACE
- REORG INDEX

maximum-days is an input parameter of type INTEGER.

image-copy-type
Specifies the types of image copies about which DSNACCAV should give you information. This value can be specified if *query-type* is COPY TABLESPACE. *image-copy-type* is an input parameter of type CHAR(1). The contents must be one of the following values:

B Specifies that you want information about partitions for which the most recent image copy was either a full image copy or an incremental image copy

F Specifies that you want information about partitions for which the most recent image copy was a full image copy

I Specifies that you want information about partitions for which the most recent image copy was an incremental image copy

maximum-extents
Specifies the maximum number of extents that a table space or index partition should use. This value can be specified if *query-type* is one of the following values:

- REORG TABLESPACE
- REORG INDEX
- EXTENTS TABLESPACE
- EXTENTS INDEX

maximum-extents is an input parameter of type INTEGER.

return-code
Specifies the return code from the DSNACCAV call, which is one of the following values:

0 DSNCCAV executed successfully.

12 An error occurred during DSNCCAV execution.

return-code is an output parameter of type INTEGER.

error-statement
If *return-code* is not 0, specifies the SQL statement or DB2 command that DB2 was executing when the error occurred. *error-statement* is an output parameter of type VARCHAR(8012).

ifi-return-code
When *query-type* is RESTRICT TABLESPACE, RESTRICT INDEX, COPY TABLESPACE, or REORG TABLESPACE, specifies the return code from the IFI call that submitted a DISPLAY DATABASE command to obtain information about restricted objects. *ifi-return-code* is an output parameter of type INTEGER.

ifi-reason-code
When *query-type* is RESTRICT TABLESPACE, RESTRICT INDEX, COPY TABLESPACE, or REORG TABLESPACE, specifies the reason code from the IFI call that submitted a DISPLAY DATABASE command to obtain information

about restricted objects. *ifi-reason-code* is an output parameter of type INTEGER.

excess-bytes
When *query-type* is RESTRICT TABLESPACE, RESTRICT INDEX, COPY TABLESPACE, or REORG TABLESPACE, specifies the number of bytes that did not fit in the return area for the IFI call that submitted a DISPLAY DATABASE command to obtain information about restricted objects. *excess-bytes* is an output parameter of type INTEGER.

message-text
If an SQL error occurs while DSNACCAV executes, contains information about the error, including the formatted SQLCA. The message text consists of from one to ten lines, each with a length of 121 bytes. the last byte of each line is a new-line character. *message-text* is an output parameter of type VARCHAR(1210).

Example of DSNACCAV invocation

Suppose that you want information on table space partitions that are in a restricted status. You want information only on table spaces that are in databases whose names begin with DSNCC. The parameter declarations and DSNACCAV call looks like this:

```
#                   DCL QUERY      CHAR(20) VARYING;
#                   DCL CRITERIA   CHAR(4096) VARYING;
#                   DCL NUMDAYS    FIXED BIN(31);
#                   DCL OPTYPE     CHAR(1) VARYING;
#                   DCL EXTENTS    FIXED BIN(31);
#                   DCL RC         FIXED BIN(31);
#                   DCL STMT       CHAR(8012) VARYING;
#                   DCL IFIRC      FIXED BIN(31);
#                   DCL IFIREASON  FIXED BIN(31);
#                   DCL IFIEXCESS  FIXED BIN(31);
#                   DCL STMT       CHAR(8012) VARYING;
#                   DCL MSGTEXT    CHAR(1331) VARYING;
#                   DCL IND1 FIXED BIN(15);
#                   DCL IND2 FIXED BIN(15);
#                   DCL IND3 FIXED BIN(15);
#                   DCL IND4 FIXED BIN(15);
#                   DCL IND5 FIXED BIN(15);
#                   DCL IND6 FIXED BIN(15);
#                   DCL IND7 FIXED BIN(15);
#                   DCL IND8 FIXED BIN(15);
#                   DCL IND9 FIXED BIN(15);
#                   DCL IND10 FIXED BIN(15);
#                   DCL IND11 FIXED BIN(15);
#                   DCL CMDMSG  SQL TYPE IS RESULT_SET_LOCATOR VARYING;
#                   DCL TSTABLE SQL TYPE IS RESULT_SET_LOCATOR VARYING;

#                   QUERY='RESTRICT TABLESPACE';
#                   IND1=0;
#                   Criteria='A.DBNAME LIKE ''DSNCC%''';
#                   IND2=0;
#                   numdays=0;
#                   IND3=0;
#                   optype='';
#                   IND4=0;
#                   extents=0;
#                   IND5=0;

#                   EXEC SQL CALL SYSPROC.DSNACCAV(:QUERY :IND1, :CRITERIA :IND2,
#                    :NUMDAYS :IND3, :OPTYPE :IND4, :EXTENTS :IND5,
#                    :RC :IND6, :STMT :IND7, :IFIRC :IND8,
#                    :IFIREASON :IND9, :IFIEXCESS :IND10, :MSGTEXT :IND11);
```

DSNACCAV output

In addition to the output parameters described in “DSNACCAV option descriptions” on page X-26, DSNACCAV returns two result sets.

The first result set contains the text from commands that DB2 executes, formatted into 80-byte records. Table 84 on page X-31 shows the format of the first result set.

The second result set contains partition information. The format of the second result set varies, depending on whether you request table space or index information. Table 85 on page X-31 shows the columns of a result set row and the DB2 catalog table that is the source of information for each column for table space queries. Table 87 on page X-34 shows the same information for index queries.

The number of rows that are returned in the second result set varies with *query-type*. Table 89 on page X-35 shows the number and types of rows that are returned from an invocation of DSNACCAV for each *query-type*.

Table 84. Result set row for DSNACCAV command output

Column name	Data type	Contents
RS_SEQUENCE	INTEGER	Sequence number of the output line
RS_DATA	CHAR(80)	A line of command output

Table 85 (Page 1 of 2). Result set row for DSNACCAV table space queries

Column name	Data type	DB2 catalog table that is the data source
NAME	CHAR(8)	SYSTABLESPACE
CREATOR	CHAR(8)	SYSTABLESPACE
BPOOL	CHAR(8)	SYSTABLESPACE
LOCKRULE	CHAR(1)	SYSTABLESPACE
LOCKMAX	INTEGER	SYSTABLESPACE
CLOSERULE	CHAR(1)	SYSTABLESPACE
ENCODING_SCHEME	CHAR(1)	SYSTABLESPACE
LOCKPART	CHAR(1)	SYSTABLESPACE
MAXROWS	SMALLINT	SYSTABLESPACE
PARTITIONS	SMALLINT	SYSTABLESPACE
TYPE	CHAR(1)	SYSTABLESPACE
SEGSIZE	SMALLINT	SYSTABLESPACE
SPACE	INTEGER	SYSTABLESPACE
NTABLES	SMALLINT	SYSTABLESPACE
STATUS	CHAR(1)	SYSTABLESPACE
STATSTIME	TIMESTAMP	SYSTABLESPACE
ERASERULE	CHAR(1)	SYSTABLESPACE
DBNAME	CHAR(8)	SYSTABLESPACE
DSETPASS	CHAR(8)	SYSTABLESPACE
Reserved	CHAR(1)	
Reserved	INTEGER	
PARTITION	SMALLINT	SYSTABLEPART
OPERATIONTIME	TIMESTAMP	SYSCOPY or SYSTABLEPART[1]
DAYS	INTEGER	SYSCOPY or SYSTABLEPART[2]
PERCOFFPOS	SMALLINT	SYSINDEXPART[3]
PERCINDREF	SMALLINT	SYSTABLEPART[4]
PERCDROP	SMALLINT	SYSTABLEPART
EXTENTS	INTEGER	None[5]
REASON	CHAR(18)	None[6]

Table 85 (Page 2 of 2). Result set row for DSNACCAV table space queries

Column name	Data type	DB2 catalog table that is the data source

Notes:

1. If *query-type* is COPY TABLESPACE or REORG TABLESPACE, the value of OPERATIONTIME is the value of the TIMESTAMP column in SYSIBM.SYSCOPY.

   If *query-type* is RUNSTATS TABLESPACE, the value of OPERATIONTIME is the value of the TIMESTAMP column in SYSIBM.SYSTABLEPART.
2. DAYS is the number of days since the last invocation of the utility. This column is derived from the OPERATIONTIME column.
3. PERCOFFPOS=(NEAROFFPOSF+FAROFFPOSF)*100/CARDF
4. PERCINDREF=(NEARINDREF+FARINDREF)*100/CARD
5. EXTENTS is the number of data set extents that the partition is using.
6. REASON is the reason that the row is in the result set. See Table 86 on page X-33 for values of REASON for each value of *query-type*.

Table 86. Values of the REASON result set column for table space queries

query-type	REASON value	REASON meaning
COPY TABLESPACE	Status from DISPLAY DATABASE command output	Table space is in restricted status COPY
	DAYS	Days since last COPY exceeds *maximum-days* value
RESTRICT TABLESPACE	Status from DISPLAY DATABASE command output	Table space is in restricted status
RUNSTATS TABLESPACE	LOAD	LOAD was run after RUNSTATS
	REORG	REORG was run after RUNSTATS
	RECOVER	RECOVER was run after RUNSTATS
	DAYS	Days since last RUNSTATS exceeds *maximum-days* value

REORG TABLESPACE	LIMIT	One of the following reasons:
		• A clustering index meets this condition:
		`((NEAROFFPOSF+FAROFFPOSF)*100/CARDF)>10`
		• A partition meets either of these conditions:
		`((NEARINDREF+FARINDREF)*100/CARD)>10`
		`PERCDROP>10`

	DAYS	Days since last REORG exceeds *maximum-days* value
	EXTENTS	Number of partition extents exceeds *maximum-extents* value
	Status from DISPLAY DATABASE command output	The table space is in restricted status REORP
EXTENTS TABLESPACE	EXTENTS	Number of partition extents exceeds *maximum-extents* value

Table 87. Result set row for DSNACCAV index queries

Column name	Data type	DB2 catalog table that is the data source
CREATOR	CHAR(8)	SYSINDEXES
NAME	VARCHAR(18)	SYSINDEXES
TBCREATOR	CHAR(8)	SYSINDEXES
TBNAME	VARCHAR(18)	SYSINDEXES
UNIQUERULE	CHAR(1)	SYSINDEXES
INDEXTYPE	CHAR(1)	SYSINDEXES
INDEXSPACE	CHAR(8)	SYSINDEXES
CLUSTERING	CHAR(1)	SYSINDEXES
ERASERULE	CHAR(1)	SYSINDEXES
CLOSERULE	CHAR(1)	SYSINDEXES
COLCOUNT	SMALLINT	SYSINDEXES
DBID	SMALLINT	SYSINDEXES
DBNAME	CHAR(8)	SYSINDEXES
BPOOL	CHAR(8)	SYSINDEXES
PGSIZE	SMALLINT	SYSINDEXES
DSETPASS	CHAR(8)	SYSINDEXES
PIECESIZE	SMALLINT	SYSINDEXES
Reserved	CHAR(1)	
PARTITIONS	SMALLINT	SYSINDEXPART[1]
PARTITION	SMALLINT	SYSINDEXPART
OPERATIONTIME	TIMESTAMP	SYSCOPY or SYSINDEXPART[2]
DAYS	INTEGER	SYSCOPY or SYSTABLEPART[3]
LEAFDIST	INTEGER	SYSINDEXPART
EXTENTS	INTEGER	None[4]
REASON	CHAR(18)	None[5]

Notes:

1. PARTITIONS is derived from the PARTITION column through this SELECT statement:

   ```
   SELECT IXNAME,IXCREATOR,MAX(PARTITION) AS PARTITIONS
     FROM SYSIBM.SYSINDEXPART
     GROUP BY IXNAME,IXCREATOR;
   ```

2. If *query-type* is REORG INDEX, the value of OPERATIONTIME is the value of the TIMESTAMP column in SYSIBM.SYSCOPY.

   If *query-type* is RUNSTATS INDEX, the value of OPERATIONTIME is the value of the TIMESTAMP column in SYSIBM.SYSINDEXPART.

3. DAYS is the number of days since the last invocation of the utility. This column is derived from the OPERATIONTIME column.

4. EXTENTS is the number of data set extents that the partition is using.

5. REASON is the reason that the row is in the result set. See Table 88 on page X-35 for values of REASON for each value of *query-type*.

Table 88. Values of the REASON result set column for index queries

query-type	**REASON value**	**REASON meaning**
RESTRICT INDEX	Status from DISPLAY DATABASE command output	Index is in restricted status
RUNSTATS INDEX	TABLESPACE LOAD	LOAD was run on the associated table space after RUNSTATS
	TABLESPACE REORG	REORG was run on the associated table space after RUNSTATS
	TABLESPACE RECOVER	RECOVER was run on the associated table space after RUNSTATS
	DAYS	Days since last RUNSTATS exceeds *maximum-days* value
REORG INDEX	LIMIT	LEAFDIST exceeds the recommended limit of 200
	DAYS	Days since last REORG exceeds *maximum-days* value
	EXTENTS	Number of partition extents exceeds *maximum-extents* value
EXTENTS INDEX	EXTENTS	Number of partition extents exceeds *maximum-extents* value

Table 89 (Page 1 of 2). Rows of the second DSNACCAV result set for each query type

query-type	**Rows returned**
COPY TABLESPACE	One row for: • Each tablespace partition that has not been copied within the number of days specified by the *maximum-days* parameter • The most recent copy of each data set in a nonpartitioned tablespace • Each table space partition that is in COPY pending status
RESTRICT TABLESPACE	One row for each table space in the subsystem that meets the criteria specified by the *search-criteria* parameter and is in a restricted status
RESTRICT INDEX	One row for each index in the subsystem that meets the criteria specified by the *search-criteria* parameter and is in a restricted status
RUNSTATS TABLESPACE	One row for: • Each tablespace partition on which LOAD, REORG, or RECOVER was run after the last time RUNSTATS was run • Each table space partition on which RUNSTATS was not run within the number of days specified by the *maximum-days* parameter

Table 89 (Page 2 of 2). Rows of the second DSNACCAV result set for each query type

query-type	**Rows returned**
RUNSTATS INDEX	One row for: • Each index partition that is defined on a table on which LOAD, REORG, or RECOVER was run after the last time RUNSTATS was run • Each index partition on which RUNSTATS was not run within the number of days specified by the *maximum-days* parameter
REORG TABLESPACE	One row for: • Each table space partition for which the number of data set extents exceeds the value specified by *maximum-extents* • Each table space partition for which the clustering index that is associated with the table has (NEAROFFPOSF+FAROFFPOSF)*100/CARDF>10 • Each table space partition for which (NEARINDREF+FARINDREF)*100/CARD>10 • Each table space partition for which PERCDROP>10 • Each table space partition on which REORG was not run within the number of days specified by the *maximum-days* parameter
REORG INDEX	One row for: • Each index partition for which LEAFDIST>200 • Each index partition for which the number of data set extents exceeds the value specified by *maximum-extents* • Each index partition on which REORG was not run within the number of days specified by the *maximum-days* parameter
EXTENTS TABLESPACE	One row for each table space partition for which the number of data set extents exceeds the value specified by *maximum-extents*
EXTENTS INDEX	One row for each index partition for which the number of data set extents exceeds the value specified by *maximum-extents*

To obtain the information from the result sets, you can write your client program to retrieve information from two result sets with known contents. However, for greater flexibility, you might want to write your client program to retrieve data from an unknown number of result sets with unknown contents. Both techniques are shown in Section 6 of *Application Programming and SQL Guide*.

Glossary and Bibliography

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 *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 *Messages and Codes*.

abnormal end of task (abend). Termination of a task, a job, or a subsystem because of an error condition that cannot be resolved during execution by recovery facilities.

access method services. A utility program that defines and manages VSAM data sets (or files).

access path. The path used to get to data specified in SQL statements. An access path can involve an index or a sequential search.

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 will fit on the active log.

address space. A range of virtual storage pages identified by a number (ASID) and a collection of segment and page tables which 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 containing a task connected to DB2 has exactly one address space connection, even though more than one task control block (TCB) can be present. See *allied address space* and *task control block*.

alias. An alternate 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 external to DB2 that is connected to DB2 and is therefore capable of requesting DB2 services.

allied thread. A thread originating at the local DB2 subsystem that may access data at a remote DB2 subsystem.

ambiguous cursor. A database cursor that is not defined with either the clauses FOR FETCH ONLY or FOR UPDATE OF, 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 SQL statements PREPARE or EXECUTE IMMEDIATE.

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 used to define DB2 to VTAM as an application program using SNA LU 6.2 protocols.

application. A program or set of programs that perform a task; for example, a payroll application.

application plan. The control structure produced during the bind process and used by DB2 to process SQL statements encountered 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 program interface (API). A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or licensed program.

application server. 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 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.

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. A table created by the SQL CREATE TABLE statement that is used to hold persistent data. Contrast with *result table* and *temporary table*.

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.

bind. The process by which the output from the DB2 precompiler is converted to a usable control structure 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. Contrast with *dynamic bind*.

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 reserved to satisfy the buffering requirements for one or more table spaces or indexes.

built-in function. Scalar function or column function.

C

CAF. Call attachment facility.

call attachment facility (CAF). A DB2 attachment facility for application programs running in TSO or MVS batch. The CAF is an alternative to the DSN command processor and allows greater control over the execution environment.

cascade delete. The enforcement of referential constraints by DB2 when it deletes all descendent rows of a deleted parent row.

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. See *communications database*.

character set. A defined set of characters.

character string. A sequence of bytes representing 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.

check constraint. See *table check constraint*.

check integrity. The condition that exists when each row in a table conforms to the table check constraints defined on that table. Maintaining check integrity requires enforcing 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 it can contain rows that violate referential constraints, table check constraints, or both.

checkpoint. A point at which DB2 records internal status information on the DB2 log that would be used in the recovery process if DB2 should abend.

CI. Control interval.

CICS. Represents (in this publication) CICS/MVS and CICS/ESA.

> **CICS/MVS**: Customer Information Control System/Multiple Virtual Storage.
> **CICS/ESA**: Customer Information Control System/Enterprise Systems Architecture.

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. To register to DB2 that an object is being accessed. This registration is also called a claim. A claim is used to ensure that an object cannot be drained until a commit is reached. Contrast with *drain*.

claim class. A specific type of object access which 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.

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. 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 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. The existence of a communication path between two partner LUs that allows information to be exchanged (for example, two DB2s connected and communicating by way of a conversation).

connection ID. An identifier supplied by the attachment facility that is associated with a specific address space connection.

consistency token. A timestamp 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, *uniqueness constraint*, and *table check 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 four bytes at the end of each control interval; it describes the free space, if any, in the control interval.

conversation. (1) A VTAM term for a dialog between two application processes, on different DB2 subsystems, that is specified by a particular *session name*, *mode name*, and *LU name*. (2) An LU 6.2 security option which allows DB2 to require the user's authorization ID and password when allocating a conversation to a partner DB2. The user is validated by the partner DB2.

coordinator. The system component that coordinates the commit or rollback of a unit of work that includes work done on one or more other systems.

correlation ID. An identifier 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.

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 cursor table is the copy of the skeleton cursor table 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 is a descendent of the last. 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 accessing data at the local subsystem on behalf of a remote subsystem.

database administrator (DBA). An individual responsible for the design, development, operation, safeguarding, maintenance, and use of a database.

database descriptor (DBD). An internal representation of DB2 database definition which reflects the data definition found in the DB2 catalog. The objects 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 access to the data stored within it.

database request module (DBRM). A data set member created by the DB2 precompiler 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 retrieved into a host variable in your program is a copy of data in the base table.

data definition name (DD name). 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.

DBCS. Double-byte character set.

DBD. Database descriptor.

DBID. Database identifier.

DBMS. Database management system.

DBRM. Database request module.

DB2 catalog. Tables maintained by DB2 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.

DB2I. DATABASE 2 Interactive.

DB2 private protocol access. A method of accessing distributed data by which you can direct a query to another DB2 system by using an alias or a three-part name to identify the DB2 subsystems at which the statements are executed. Contrast with *DRDA access*.

DB2 private protocol connection. A DB2 private connection of the application process. See also *private connection*.

DCLGEN. Declarations generator.

DDF. Distributed data facility.

DD name. 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.

default value. A predetermined value, attribute, or option that is assumed when no other is explicitly specified.

delimited identifier. A sequence of characters enclosed within 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) is a dependent if it 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 dependent in at least one referential constraint.

descendent. An object is a descendent of another object if it is a dependent of the object, or if it is the dependent of a descendent of that object.

descendent row. A row that is dependent on another row or a row that is a dependent of a descendent row.

descendent table. A table that is a dependent of another table or a dependent of a descendent table.

direct access storage device (DASD). A device in which access time is independent of the location of the data.

directory. The system database that contains internal objects such as database descriptors and skeleton cursor tables.

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. The IMS data manipulation language; a common high-level interface between a user application and IMS.

double-byte character set (DBCS). A set of characters used by national languages such as Japanese and Chinese that have more symbols than can be represented by a single byte. Each character is two bytes in length, and therefore requires special hardware to be displayed or printed.

double-precision floating point number. A 64-bit approximate representation of a real number.

drain. To acquire a locked resource by quiescing access to that object.

drain lock. A lock on a claim class which prevents a claim from occurring.

DRDA. Distributed relational database architecture.

DRDA access. A method of accessing distributed data by which you can explicitly connect to another location, using an SQL statement, to execute packages that have been previously bound at that location. The SQL CONNECT statement is used to identify application servers, and SQL statements are executed using packages that were previously bound at those servers. Contrast with *DB2 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 used to represent character data in the MVS, VM, VSE, and OS/400 environments. Contrast with *ASCII*.

EDM pool. A pool of main storage used for database descriptors and application plans.

embedded SQL. SQL statements coded within an application program. See *static SQL*.

escape character. The symbol used to enclose an SQL delimited identifier. The escape character is the quotation mark ("), except in COBOL applications, where the symbol (either a quotation mark or an apostrophe) can be assigned by the user.

ESDS. Entry sequenced data set.

EUR. IBM European Standards.

exception table. A table that holds rows that violate referential constraints or table check constraints found by the CHECK DATA utility.

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.

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 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 unused space in a page, that is, the space not used to store records or control information.

function. A scalar function or column function. Same as *built-in function*.

G

GB. Gigabyte (1,073,741,824 bytes).

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 regarding attempts 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. A coupling facility cache structure 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 written in a host language that contains embedded SQL statements.

HSM. Hierarchical storage manager.

I

IDCAMS. An IBM program used to process access method services (AMS) 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 separate from DB2 issues via the MVS subsystem interface to inform DB2 of its existence and initiate the process of becoming connected to DB2.

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 will continue 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 used to determine the order of index entries.

index partition. A VSAM data set that is contained within a partitioned index space.

index space. A page set 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 this unit of recovery is to be committed or rolled back. At emergency restart, if DB2 does not have the information needed to make this decision, its unit of recovery is *indoubt* until DB2 obtains this information from the coordinator.

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 when it is restarted. These units of recovery are termed *inflight*.

inline copy. A copy produced by the LOAD or REORG utility. The data set produced by the inline copy is logically equivalent to a full image copy produced by running the COPY utility with read-only access (SHRLEVEL REF).

instrumentation facility component identifier (IFCID). Names a traceable event and identifies the trace record of that event. As a parameter on the -START TRACE and -MODIFY TRACE commands, it specifies tracing the corresponding event.

Interactive System Productivity Facility (ISPF). An IBM licensed program that provides interactive dialog services.

internal resource lock manager (IRLM). An MVS subsystem used by DB2 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*, *repeatable read*, *uncommitted read*, and *read stability*.

ISPF. Interactive System Productivity Facility.

ISPF/PDF. Interactive System Productivity Facility/Program Development Facility.

J

JCL. Job control language.

JES. MVS Job Entry Subsystem.

JIS. Japanese Industrial Standard.

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.

KSDS. Key sequenced data set.

L

labeled duration. A number that represents a duration of years, months, days, hours, minutes, seconds, or microseconds.

leaf page. A page that contains pairs of keys and RIDs and that points to actual data. Contrast with *nonleaf page*.

link-edit. To create a loadable computer program using a linkage editor.

L-lock. See *logical lock*.

load module. A program unit that is suitable for loading into main storage for execution. The output of a linkage editor.

local. Refers to any object maintained by the local DB2 subsystem. A *local table*, for example, is a table 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 or page lock to a table space lock because the number of page locks 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 concurrently running programs can have to a resource held by a DB2 lock.

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 table, or a table space.

log. A collection of records that describe the events that occur during DB2 execution and 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. The lock type used by transactions to control intra- and inter-DB2 data concurrency between transactions.

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). In IMS, the processing that 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 DB2 generates and associates with each log record. DB2 also uses the LRSN for page versioning. The

LRSNs generated by a given DB2 data sharing group 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 will be written.

LRH. Log record header.

LRSN. See *log record sequence number*.

LU name. From *logical unit name*, 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 used by the REORG utility to map between the RIDs of data records in the original copy and 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. 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

NID (network identifier). The network ID assigned by IMS or CICS, or if the connection type is RRSAF, the OS/390 RRS Unit of Recovery ID (URID).

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 assigned sequentially to each IMS schedule since the last cold start of IMS and used as an identifier for a unit of work. In an 8-byte format, the first four bytes contain the schedule number and the last four 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.

P

package. Also *application package*. An object containing a set of SQL statements that have been bound statically and that are available for processing.

package list. An ordered list of package names that may be used to extend an application plan.

package name. The name given an object 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 (4KB or 32KB) or index space (4KB). In a table space, a page contains one or more rows of a table.

page set. A table space or index space consisting of pages that are either 4KB or 32KB in size. Each page set is made from a collection of VSAM data sets.

page set recovery pending (PSRCP). A restrictive state of an index space in which the page set is in a recovery pending state. 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. 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 gigabytes, depending upon 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 subdivided into parts (based upon index key range), each of which may be processed by utilities independently.

partner logical unit. An access point in the SNA network that is connected to the local DB2 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 an application accesses is consistent with other data. 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.

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 recorded for that ID in the DB2 catalog.

process. A general term for a unit that depends on the environment, but has the same basic properties in every environment. A process involves the execution of one or more programs, and is the unit to which resources and locks are allocated. The execution of an SQL statement is always associated with some process.

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. OS/VS2 MVS 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, known as *phantom rows*, that were inserted and committed by a concurrently executing application process.

rebind. To create a new application plan for an application program that has been bound previously. If, for example, you have added an index for a table accessed by your application, 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) pool. An area of main storage above the 16MB 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 their sequence. The information recorded is used for recovery in the event of a failure during DB2 execution.

recovery pending (RECP). This condition prevents SQL access to a table space or index space that may need to be recovered.

RECP. Recovery pending.

redo. A state of a unit of recovery which indicates that changes made 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 enforcing 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 allocated to the data set or file to which it belongs.

remote. Refers to any object maintained by a remote DB2 subsystem; that is, by a DB2 subsystem other than the local one. A *remote view*, for instance, is a view 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 may even operate on the same processor under the same MVS system.

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 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.

resource limit specification table. A site-defined table that specifies the limits to be enforced by the resource limit facility.

result table. The set of rows specified by a SELECT statement.

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*.

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 enclosed in parentheses. See also *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. Also *application server (AS)*. The target for a request from a remote RDBMS, the RDBMS that provides the data.

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.

shift-in character. A special control character (X'0F') used in EBCDIC systems to denote that the following bytes represent SBCS characters. See *shift-out character*.

shift-out character. A special control character (X'0E') used in EBCDIC systems to denote that the following bytes, up to the next shift-in control character, represent DBCS characters.

sign-on. A request made on behalf of an individual CICS or IMS application process by an attach 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.

SMF. System management facility.

SMS. Storage Management Subsystem.

SNA. Systems Network Architecture.

source program. A set of host language statements and SQL statements that is processed by an SQL precompiler.

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. 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 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 used to enclose an SQL delimited identifier. This symbol is the quotation mark ("). See *escape character*.

SQL return code. Either SQLCODE or SQLSTATE.

SQL string delimiter. A symbol used to enclose an SQL string constant. The SQL string delimiter is the apostrophe ('), except in COBOL applications, in which case the symbol (either an apostrophe or a quotation mark) may be assigned by the user.

SQLCA. SQL communication area.

SQLDA. SQL descriptor area.

SQL/DS. SQL/Data System. Also known as *DB2/VSE & VM*.

SSI. MVS subsystem interface.

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 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 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 having the second highest level of authority within DB2. System administrators make decisions about how DB2 is to be used and implement those decisions by choosing system parameters. They monitor the system and change its characteristics to meet changing requirements and new data processing goals.

system agent. A work request that DB2 creates internally.

system conversation. The conversation that two DB2s 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. Synonymous with *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 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 because each of them contains a table that is a parent or descendent of a table in one of the others.

task control block (TCB). A control block 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 *address space connection*.

TCB. MVS task control block.

temporary table. A table created by the SQL CREATE GLOBAL TEMPORARY TABLE statement that is used to hold temporary data. Contrast with *result table* and *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). Provides interactive time sharing from remote terminals.

timestamp. A seven-part value that consists of a date and time 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 IMSenvironments 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*.

type 2 indexes. A new type of indexes available in Version 4. They differ from *type 1 indexes* in several respects; for example, they are the only indexes allowed on a table space that uses *row locks*.

U

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 made by the unit of recovery to recoverable DB2 resources must be backed out.

unique index. An index which ensures that no identical key values are stored in a table.

uniqueness constraint. The rule that no two values in a primary key or 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*.

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.

UT. Utility-only access.

V

value. The smallest unit of data 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, or packages.

> **A version of a program** is the source code 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 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.

view. An alternative representation of data from one or more tables. A view can include all or some of the columns contained in tables on which it is defined.

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. MVS Virtual telecommunication access method.

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 for OS/390 Version 5

- *Administration Guide, SC26-8957*
- *Application Programming and SQL Guide, SC26-8958*
- *Call Level Interface Guide and Reference, SC26-8959*
- *Command Reference, SC26-8960*
- *Data Sharing: Planning and Administration, SC26-8961*
- *Data Sharing Quick Reference Card, SX26-3841*
- *Diagnosis Guide and Reference, LY27-9659*
- *Diagnostic Quick Reference Card, LY27-9660*
- *Installation Guide, GC26-8970*
- *Application Programming Guide and Reference for Java™, SC26-9547*
- *Licensed Program Specifications, GC26-8969*
- *Messages and Codes, GC26-8979*
- *Reference for Remote DRDA Requesters and Servers, SC26-8964*
- *Reference Summary, SX26-3842*
- *Release Guide, SC26-8965*
- *SQL Reference, SC26-8966*
- *Utility Guide and Reference, SC26-8967*
- *What's New?, GC26-8971*
- *Program Directory*

DB2 PM for OS/390 Version 5

- *Batch User's Guide, SC26-8991*
- *Command Reference, SC26-8987*
- *General Information, GC26-8982*
- *Getting Started on the Workstation, SC26-8989*
- *Master Index, SC26-8984*
- *Messages Manual, SC26-8988*
- *Online Monitor User's Guide, SC26-8990*
- *Report Reference Volume 1, SC26-8985*
- *Report Reference Volume 2, SC26-8986*
- *Program Directory*

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*

C/370

- *IBM SAA AD/Cycle C/370 Programming Guide, SC09-1356*
- *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, SC09-2190*

CICS/ESA

- *CICS/ESA Application Programming Guide, SC33-1169*
- *CICS/ESA Application Programming Reference, SC33-1170*
- *CICS/ESA CICS - RACF Security Guide, SC33-1185*
- *CICS/ESA CICS-Supplied Transactions, SC33-1168*
- *CICS/ESA Customization Guide, SC33-1165*
- *CICS/ESA Data Areas, LY33-6083*
- *CICS/ESA Installation Guide, SC33-1163*
- *CICS/ESA Intercommunication Guide, SC33-1181*
- *CICS/ESA Messages and Codes, SC33-1177*
- *CICS/ESA Operations and Utilities Guide, SC33-1167*
- *CICS/ESA Performance Guide, SC33-1183*
- *CICS/ESA Problem Determination Guide, SC33-1176*
- *CICS/ESA Resource Definition Guide, SC33-1166*
- *CICS/ESA System Definition Guide, SC33-1164*
- *CICS/ESA System Programming Reference, GC33-1171*

CICS/MVS

- *CICS/MVS Application Programming Primer, SC33-0139*
- *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 or OS/390

- *IBM C/C++ for MVS/ESA Library Reference, SC09-1995*
- *IBM C/C++ for MVS/ESA Programming Guide, SC09-1994*
- *IBM C/C++ for OS/390 User's Guide, SC09-2361*

IBM COBOL for MVS & VM

- *IBM COBOL for MVS & VM Language Reference, SC26-4769*
- *IBM COBOL for MVS & VM Programming Guide, SC26-4767*

Conversion Guides

- *DBMS Conversion Guide: DATACOM/DB to DB2, GH20-7564*
- *DBMS Conversion Guide: IDMS to DB2, GH20-7562*
- *DBMS Conversion Guide: Model 204 to DB2 or SQL/DS, GH20-7565*
- *DBMS Conversion Guide: VSAM to DB2, GH20-7566*
- *IMS-DB and DB2 Migration and Coexistence Guide, GH21-1083*

Cooperative Development Environment

- *CoOperative Development Environment/370: Debug Tool, SC09-1623*

DATABASE 2 for Common Servers

- *DATABASE 2 Administration Guide for common servers, S20H-4580*
- *DATABASE 2 Application Programming Guide for common servers, S20H-4643*
- *DATABASE 2 Software Developer's Kit for AIX: Building Your Applications, S20H-4780*
- *DATABASE 2 Software Developer's Kit for OS/2: Building Your Applications, S20H-4787*
- *DATABASE 2 SQL Reference for common servers, S20H-4665*
- *DATABASE 2 Call Level Interface Guide and Reference for common servers, S20H-4644*

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*

DataPropagator Relational

- *DataPropagator Relational User's Guide, SC26-3399*
- *IBM An Introduction to DataPropagator Relational, GC26-3398*

Data Facility Data Set Services

- *Data Facility Data Set Services: User's Guide and Reference, SC26-4125*

Database Design

- *DB2 Database Design and Implementation Using DB2, SH24-6101*
- *DB2 Design and Development Guide, Gabrielle Wiorkowski and David Kull, Addison Wesley*
- *Handbook of Relational Database Design, C. Fleming and B Von Halle, Addison Wesley*
- *Principles of Database Systems, Jeffrey D. Ullman, Computer Science Press*

DataHub

- *IBM DataHub General Information, GC26-4874*

DB2 Universal Database

- *DB2 Universal Database Administration Guide, S10J-8157*
- *DB2 Universal Database API Reference, S10J-8167*
- *DB2 Universal Database Building Applications for UNIX Environments, S10J-8161*
- *DB2 Universal Database Building Applications for Windows and OS/2 Environments, S10J-8160*
- *DB2 Universal Database CLI Guide and Reference, S10J-8159*
- *DB2 Universal Database SQL Reference, S10J-8165*

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: 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 for Data Sets, SC26-4921*
- *DFSMS/MVS: Utilities, SC26-4926*
- *MVS/DFP: Managing Non-VSAM Data Sets, SC26-4557*

DFSORT

- *DFSORT Application Programming: Guide, SC33-4035*

Distributed Relational Database

- *Data Stream and OPA Reference, SC31-6806*
- *Distributed Relational Database Architecture: Application Programming Guide, SC26-4773*
- *Distributed Relational Database Architecture: Connectivity Guide, SC26-4783*
- *Distributed Relational Database Architecture: Evaluation and Planning Guide, SC26-4650*
- *Distributed Relational Database Architecture: Problem Determination Guide, SC26-4782*
- *Distributed Relational Database: Every Manager's Guide, GC26-3195*
- *IBM SQL Reference, SC26-8416*
- *Open Group Technical Standard (the Open Group presently makes the following books available through their website at www.opengroup.org):*
  - *DRDA Volume 1: Distributed Relational Database Architecture (DRDA), ISBN 1-85912-295-7*
  - *DRDA Volume 3: Distributed Database Management (DDM) Architecture, ISBN 1-85912-206-X*

Education

- *Dictionary of Computing, SC20-1699*
- *IBM Enterprise Systems Training Solutions Catalog, GR28-5467*

Enterprise System/9000 and Enterprise System/3090

- *Enterprise System/9000 and Enterprise System/3090 Processor Resource/System Manager Planning Guide, GA22-7123*

FORTRAN

- *VS FORTRAN Version 2: Language and Library Reference, SC26-4221*
- *VS FORTRAN Version 2: Programming Guide for CMS and MVS, SC26-4222*

High Level Assembler

- *High Level Assembler/MVS and VM and VSE Language Reference, SC26-4940*
- *High Level Assembler/MVS and VM and VSE Programmer's Guide, SC26-4941*

Parallel Sysplex Library

- *System/390 MVS Sysplex Application Migration, GC28-1211*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28-1210*
- *System/390 MVS Sysplex Overview: An Introduction to Data Sharing and Parallelism, GC28-1208*
- *System/390 MVS Sysplex Systems Management, GC28-1209*
- *System/390 MVS 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 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 Version 4 Messages and Codes, SC34-4450*
- *ISPF Version 4 for MVS Dialog Management Guide, SC34-4213*
- *ISPF/PDF Version 4 for MVS Guide and Reference, SC34-4258*
- *ISPF and ISPF/PDF Version 4 for MVS Planning and Customization, SC34-4134*

Language Environment for MVS & VM

- *Language Environment for MVS & VM Concepts Guide, GC26-4786*
- *Language Environment for MVS & VM Debugging and Run-Time Messages Guide, SC26-4829*
- *Language Environment for MVS & VM Installation and Customization, SC26-4817*
- *Language Environment for MVS & VM Programming Guide, SC26-4818*
- *Language Environment for MVS & VM Programming Reference, SC26-3312*

MVS/ESA

- *MVS/ESA Analyzing Resource Measurement Facility Monitor I and Monitor II Reference and User's Guide, LY28-1007*
- *MVS/ESA Analyzing Resource Measurement Facility Monitor III Reference and User's Guide, LY28-1008*
- *MVS/ESA Application Development Reference: Assembler Callable Services for OpenEdition MVS, SC23-3020*
- *MVS/ESA Data Administration: Utilities, SC26-4516*
- *MVS/ESA Diagnosis: Procedures, LY28-1844*
- *MVS/ESA Diagnosis: Tools and Service Aids, LY28-1845*
- *MVS/ESA Initialization and Tuning Guide, SC28-1451*
- *MVS/ESA Initialization and Tuning Reference, SC28-1452*
- *MVS/ESA Installation Exits, SC28-1459*
- *MVS/ESA JCL Reference, GC28-1479*
- *MVS/ESA JCL User's Guide, GC28-1473*
- *MVS/ESA JES2 Initialization and Tuning Guide, SC28-1453*
- *MVS/ESA MVS Configuration Program, GC28-1615*
- *MVS/ESA Planning: Global Resource Serialization, GC28-1450*
- *MVS/ESA Planning: Operations, GC28-1441*
- *MVS/ESA Planning: Workload Management, GC28-1493*
- *MVS/ESA Programming: Assembler Services Guide, GC28-1466*
- *MVS/ESA Programming: Assembler Services Reference, GC28-1474*
- *MVS/ESA Programming: Authorized Assembler Services Guide, GC28-1467*
- *MVS/ESA Programming: Authorized Assembler Services Reference, Volumes 1-4, GC28-1475, GC28-1476, GC28-1477, GC28-1478*
- *MVS/ESA Programming: Extended Addressability Guide, GC28-1468*
- *MVS/ESA Programming: Sysplex Services Guide, GC28-1495*
- *MVS/ESA Programming: Sysplex Services Reference, GC28-1496*
- *MVS/ESA Programming: Workload Management Services, GC28-1494*
- *MVS/ESA Routing and Descriptor Codes, GC28-1487*
- *MVS/ESA Setting Up a Sysplex, GC28-1449*
- *MVS/ESA SPL: Application Development Guide, GC28-1852*
- *MVS/ESA System Codes, GC28-1486*
- *MVS/ESA System Commands, GC28-1442*
- *MVS/ESA System Management Facilities (SMF), GC28-1457*
- *MVS/ESA System Messages Volume 1, GC28-1480*
- *MVS/ESA System Messages Volume 2, GC28-1481*
- *MVS/ESA System Messages Volume 3, GC28-1482*
- *MVS/ESA Using the Subsystem Interface, SC28-1502*

Net.Data for OS/390

- *Net.Data Language Environment Guide, http://www.ibm.com/software/net.data/docs*
- *Net.Data Programming Guide, http://www.ibm.com/software/net.data/docs*
- *Net.Data Reference Guide, http://www.ibm.com/software/net.data/docs*

NetView

- *NetView Installation and Administration Guide, SC31-8043*
- *NetView User's Guide, SC31-8056*

ODBC

- *ODBC 2.0 Programmer's Reference and SDK Guide, ISBN 1-55615-658-8*
- *Inside ODBC, ISBN 1-55615-815-7*

OS/390

- *OS/390 C/C++ Programming Guide, SC09-2362*
- *OS/390 C/C++ Run-Time Library Reference, SC28-1663*
- *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: 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 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) 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 REXX Reference, SC28-1975*
- *OS/390 TSO/E User's Guide, SC28-1968*

OS/390 OpenEdition

- *OS/390 OpenEdition DCE Administration Guide, SC28-1584*
- *OS/390 OpenEdition DCE Introduction, GC28-1581*
- *OS/390 R1 OE DCE Messages and Codes, ST01-0920*
- *OS/390 OpenEdition Command Reference, SC28-1892*
- *OS/390 OpenEdition Messages and Codes, SC28-1908*
- *OS/390 OpenEdition Planning, SC28-1890*
- *OS/390 OpenEdition User's Guide, SC28-1891*

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*

Query Management Facility

- *Query Management Facility: Managing QMF for MVS, SC26-8218*
- *Query Management Facility: Reference, SC26-4716*
- *Query Management Facility: Using QMF, SC26-8078*

Remote Recovery Data Facility

- *Remote Recovery Data Facility Program Description and Operations, LY37-3710*

Resource Access Control Facility (RACF)

- *External Security Interface (RACROUTE) Macro Reference for MVS and VM, GC28-1366*
- *Resource Access Control Facility (RACF) Auditor's Guide, SC28-1342*
- *Resource Access Control Facility (RACF) Command Language Reference, SC28-0733*
- *Resource Access Control Facility (RACF) General Information Manual, GC28-0722*

- *Resource Access Control Facility (RACF) General User's Guide, SC28-1341*
- *Resource Access Control Facility (RACF) Security Administrator's Guide, SC28-1340*
- *Recource Access Control Facility (RACF) System Programmer's Guide, SC28-1343*

Storage Management

- *MVS/ESA Storage Management Library: Implementing System-Managed Storage, SC26-3123*
- *MVS/ESA Storage Management Library: Leading an Effective 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

- *IBM System/370 ESA Principles of Operation, SA22-7200*
- *IBM System/390 ESA Principles of Operation, SA22-7205*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28-1210*

System Modification Program Extended (SMP/E)

- *System Modification Program Extended (SMP/E) Reference, SC28-1107*
- *System Modification Program Extended (SMP/E) User's Guide, SC28-1302*

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*

TSO Extensions

- *TSO/E CLISTS, SC28-1876*
- *TSO/E Command Reference, SC28-1881*
- *TSO/E Customization, SC28-1872*
- *TSO/E Messages, GC28-1885*
- *TSO/E Programming Guide, SC28-1874*
- *TSO/E Programming Services, SC28-1875*
- *TSO/E User's Guide, SC28-1880*

VS COBOL II

- *VS COBOL II Application Programming Guide for MVS and CMS, SC26-4045*
- *VS COBOL II Application Programming: Language Reference, SC26-4047*
- *VS COBOL II Installation and Customization for MVS, SC26-4048*

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

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

We'd Like to Hear from You

DB2 for OS/390
Version 5
Utility Guide and Reference

Publication No. SC26-8967-02

Please use one of the following ways to send us your comments about this book:

- Mail—Use the Readers' Comments form on the next page. If you are sending the form from a country other than the United States, give it to your local IBM branch office or IBM representative for mailing.
- Fax—Use the Readers' Comments form on the next page and fax it to this U.S. number: 800-426-7773 or (408) 463-4393.
- Electronic mail—Use one of the following network IDs:
  - IBMMail: USIBMXFC @ IBMMAIL
  - IBMLink: DB2PUBS @ STLVM27
  - Internet: DB2PUBS@VNET.IBM.COM

  Be sure to include the following with your comments:
  - Title and publication number of this book
  - Your name, address, and telephone number or your name and electronic address if you would like a reply

Your comments should pertain only to the information in this book and the way the information is presented. To request additional publications, or to comment on other IBM information or the function of IBM products, please give your comments to your IBM representative or to your IBM authorized remarketer.

IBM may use or distribute your comments without obligation.

Readers' Comments

DB2 for OS/390
Version 5
Utility Guide and Reference

Publication No. SC26-8967-02

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
SC26-8967-02

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

SC26-8967-02

Cut or Fold
Along Line

Program Number: 5655-DB2

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

DB2 for OS/390
Version 5

SC26-8957	Administration Guide
SC26-8958	Application Programming and SQL Guide
SC26-8959	Call Level Interface Guide and Reference
SC26-8960	Command Reference
SC26-8961	Data Sharing: Planning and Administration
SX26-3841	Data Sharing Quick Reference
LY27-9659	Diagnosis Guide and Reference
LY27-9660	Diagnosis Quick Reference
GC26-8970	Installation Guide
GC26-8979	Master Index
SC26-8962	Messages and Codes
SC26-8964	Reference for Remote DRDA Requesters and Servers
SX26-3842	Reference Summary
SC26-8965	Release Guide
SC26-8966	SQL Reference
SC26-8967	Utility Guide and Reference
GC26-8971	What's New?

SC26-8967-02