DB2 Universal Database for OS/390 and z/OS

# Administration Guide

*Version 7*

SC26-9931-01

DB2 Universal Database for OS/390 and z/OS

# Administration Guide

*Version 7*

SC26-9931-01

**Note**

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 1095.

**Second Edition, Softcopy Only (August 2001)**

This edition applies to Version 7 of IBM DATABASE 2 Universal Database Server for OS/390 and z/OS (DB2 for OS/390 and z/OS), 5675-DB2, and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

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

This and other books in the DB2 for OS/390 and z/OS library are periodically updated with technical changes. These updates are made available to licensees of the product on CD-ROM and on the Web (currently at www.ibm.com/software/data/db2/os390/library.html). Check these resources to ensure that you are using the most current information.

# Contents

**Chapter 18. Managing the log and the bootstrap data set . . . . . . . . 331**

**Chapter 19. Restarting DB2 after termination . . . . . . . . . . . . 347**

**Chapter 20. Maintaining consistency across multiple systems . . . . . 359**

# About this book

This two-volume book provides guidance information that you can use to perform a variety of administrative tasks with DB2 Universal Database™ for OS/390® and z/OS (DB2®).

**Important**

In this version of DB2 for OS/390 and z/OS, some utility functions are available as optional products. You must separately order and purchase a license to such utilities, and discussion of those utility functions in this publication is not intended to otherwise imply that you have a license to them.

## Who should read this book

This book is primarily intended for system and database administrators. It assumes that the user is familiar with:

- The basic concepts and facilities of DB2
- The MVS Time Sharing Option (TSO) and the MVS Interactive System Productivity Facility (ISPF)
- The basic concepts of Structured Query Language (SQL)
- The basic concepts of Customer Information Control System (CICS®)
- The basic concepts of Information Management System (IMS™)
- How to define and allocate MVS data sets using MVS job control language (JCL).

Certain tasks require additional skills, such as knowledge of Virtual Telecommunications Access Method (VTAM®) to set up communication between DB2 subsystems, or knowledge of the IBM System Modification Program (SMP/E) to install IBM licensed programs.

## Product terminology and citations

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

When referring to a DB2 product other than DB2 for OS/390 and z/OS, this book uses the product’s full name to avoid ambiguity.

The following terms are used as indicated:

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

**C and C language**
Represent the C programming language.

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

**IMS** Represents IMS or IMS/ESA®.

**MVS** Represents the MVS element of OS/390.

**OS/390**
Represents the OS/390 or z/OS operating system.

**RACF®**
Represents the functions that are provided by the RACF component of the SecureWay® Security Server for OS/390 or by the RACF component of the OS/390 Security Server.

## How to send your comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 for OS/390 and z/OS documentation. You can use any of the following methods to provide comments:

- Send your comments by e-mail to db2pubs@vnet.ibm.com and include the name of the product, the version number of the product, and the number of the book. If you are commenting on specific text, please list the location of the text (for example, a chapter and section title, page number, or a help topic title).
- Send your comments from the Web. Visit the Web site at:

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

  The Web site has a feedback page that you can use to send comments.
- Complete the readers' comment form at the back of the book and return it by mail, by fax (800-426-7773 for the United States and Canada), or by giving it to an IBM representative.

# Summary of changes to this book

This section summarizes the major changes to this book for Version 7. See "Chapter 1. Summary of changes to DB2 for OS/390 and z/OS Version 7" on page 3 for an overview of the changes in Version 7 of DB2 for OS/390 and z/OS.

Part 1. Introduction has changed as follows: "Chapter 2. System planning concepts" on page 7 provides a summary of system planning concepts for DB2 for OS/390 and z/OS and provides pointers to other information sources for more detailed information. This chapter contains higher-level information than it did in previous releases, because *An Introduction to DB2 for OS/390*, new in Version 7, has extensive conceptual information about DB2 for OS/390 and z/OS.

Part 2. Designing a database: advanced topics, formerly entitled, ″Designing a database,″ has undergone a change in scope for Version 7. In prior versions, ″Designing a database″ provided a range of information, from basic to advanced, about designing a database. "Part 2. Designing a database: advanced topics" on page 27 now presents only the advanced topics. The newest member of the DB2 for OS/390 and z/OS library, *An Introduction to DB2 for OS/390*, covers basic information about designing and implementing a database. Table 6 on page 29 and Table 7 on page 41 provide roadmaps to information that was formerly part of ″Designing a database.″

Part 3. Security and auditing has changed as follows:

- "Explicit privileges and authorities" on page 104 describes the explicit Java class privileges and also the updated DBADM authority.
- "Implicit privileges of ownership" on page 114 describes the implicit Java class privileges and also the updated DBADM authority.
- "Controlling access to catalog tables for stored procedures" on page 124 has guidelines for granting access to catalog tables that programmers need to develop stored procedures.
- "The REVOKE statement" on page 146 describes how the RESTRICT clause of the REVOKE statment applies to Jars (Java classes for a routine).
- "Controlling requests from remote applications" on page 176 explains Kerberos security tickets, encrypted user IDs and encrypted passwords, and encrypted changed passwords.
- "Establishing Kerberos authentication through RACF" on page 212 explains how to implement Kerberos authentication through RACF.
- "Chapter 14. Auditing" on page 219 describes how an authorization ID can be mapped to a RACF ID from a Kerberos security ticket.

Part 4. Operation and recovery has changed as follows:

- "Chapter 16. Basic operation" on page 249 adds new commands DISPLAY DDF and SET SYSPARM, and highlights new options for the commands DISPLAY LOG, SET LOG, and RECOVER POSTPONED in "DB2 operator commands" on page 250.
- "Chapter 17. Monitoring and controlling DB2 and its connections" on page 267 describes:
  - How to reset restrictions so that DB2 can execute START DATABASE ACCESS(FORCE), as described in "Starting a table space or index space that has restrictions" on page 268.

  - The sample output in “The command DISPLAY DDF” on page 309; the DETAIL report includes the number of connections that are waiting to be associated with database access threads.
  - How to modify subsystem parameter values dynamically while DB2 is running by using the SET SYSPARM command as described in “Changing subsystem parameter values” on page 329.
- “Chapter 18. Managing the log and the bootstrap data set” on page 331 describes:
  - How to cancel long running threads without backing out data changes, by using the NOBACKOUT option of the CANCEL THREAD command (see “Rolling back work” on page 332)
  - How to use either the LOGLOAD option or the CHKTIME option of the SET LOG command to dynamically change the checkpoint frequency (see “Changing the checkpoint frequency dynamically” on page 340)
- “Chapter 19. Restarting DB2 after termination” on page 347 describes:
  - How to use the UR log threshold option to inform you about long running URs (see “Normal restart and recovery” on page 348)
  - Why you might want to use the CANCEL option of the RECOVER POSTPONED command (see “Resolving postponed units of recovery” on page 355)
- “Chapter 21. Backing up and recovering databases” on page 373 describes why you might want to use the LIGHT(YES) option of the START DB2 command for some members of a data sharing environment (see “Preparing for disaster recovery” on page 385 ).
- “Chapter 22. Recovery scenarios” on page 409 describes a procedure for enlarging a data set for the work file database (see “Out of disk space or extent limit reached” on page 440).

Part 5. Performance monitoring and tuning has changed as follows:

- “Chapter 28. Improving resource utilization” on page 579 contains revised recommendations on setting address space priorities.
- “Chapter 30. Improving concurrency” on page 643 describes optimistic concurrency control for scrollable cursors, which can shorten the amount of time that locks might be held. For queries with isolation level RS or CS, the chapter also explains why you might want to use an installation option that indicates if predicate evaluation can occur on the uncommitted data of other transactions, which can reduce the number of locks that are acquired.
- “Chapter 31. Tuning your queries” on page 711 contains recommendations on using scrollable cursors efficiently.
- “Chapter 32. Maintaining statistics in the catalog” on page 765 has information about the new DB2 catalog tables for history statistics. The chapter also explains how to use the new catalog columns LEAFNEAR and LEAFFAR to determine when an index should be reorganized.
- “Chapter 33. Using EXPLAIN to improve SQL performance” on page 789 contains information about views and table expressions that are defined with UNION and UNION ALL operators.
- “Chapter 35. Tuning and monitoring in a distributed environment” on page 857 explains how block fetch works for scrollable cursors. The chapter also describes how to use the FETCH FIRST n ROWS ONLY clause of the SELECT statement to limit the number of rows that DB2 prefetches to a specific number for a distributed query that uses DRDA access.

Appendix B. Writing exit routines has changed as follows:

- “Connection and sign-on routines” on page 901 has information on using the USER and USING keywords on the CONNECT statement.
- “Access control authorization exit” on page 909 has information on function resolution during an AUTOBIND. Also, the parameter list for the access control authorization routine has been updated for Jars (Java classes for a routine).
- “Exception processing” on page 920 explains how the EXPLRC1 value affects DB2 processing.
- “Determining if the exit routine is active” on page 921 explains how to determine whether the exit routine or DB2 is performing authorization checks.

# Part 1. Introduction

# Chapter 1. Summary of changes to DB2 for OS/390 and z/OS Version 7

DB2 for OS/390 and z/OS Version 7 delivers an enhanced relational database server solution for OS/390. This release focuses on greater ease and flexibility in managing your data, better reliability, scalability, and availability, and better integration with the DB2 family.

In Version 7, some utility functions are available as optional products; you must separately order and purchase a license to such utilities. Discussion of utility functions in this publication is not intended to otherwise imply that you have a license to them. See *DB2 Utility Guide and Reference* for more information about utilities products.

## Enhancements for managing data

Version 7 delivers the following enhancements for managing data:

- DB2 now collects a comprehensive statistics history that:
  - Lets you track changes to the physical design of DB2 objects
  - Lets DB2 predict future space requirements for table spaces and indexes more accurately and run utilities to improve performance
- Database administrators can now manage DB2 objects more easily and no longer must maintain their utility jobs (even when new objects are added) by using enhancements that let them:
  - Dynamically create object lists from a pattern-matching expression
  - Dynamically allocate the data sets that are required to process those objects
- More flexible DBADM authority lets database administrators create views for other users.
- Enhancements to management of constraints let you specify a constraint at the time you create primary or unique keys. A new restriction on the DROP INDEX statement requires that you drop the primary key, unique key, or referential constraint before you drop the index that enforces a constraint.

## Enhancements for reliability, scalability, and availability

Version 7 delivers the following enhancements for the reliability, scalability, and availability of your e-business:

- The DB2 Utilities Suite provides utilities for all of your data management tasks that are associated with the DB2 catalog.
- The new UNLOAD utility lets you unload data from a table space or an image copy data set. In most cases, the UNLOAD utility is faster than the DSNTIAUL sample program, especially when you activate partition parallelism for a large partitioned table space. UNLOAD is also easier to use than REORG UNLOAD EXTERNAL.
- The new COPYTOCOPY utility lets you make additional image copies from a primary image copy and registers those copies in the DB2 catalog. COPYTOCOPY leaves the target object in read/write access mode (UTRW), which allows Structured Query Language (SQL) statements and some utilities to run concurrently with the same target objects.

- Parallel LOAD with multiple inputs lets you easily load large amounts of data into partitioned table spaces for use in data warehouse applications or business intelligence applications. Parallel LOAD with multiple inputs runs in a single step, rather than in different jobs.
- A faster online REORG is achieved through the following enhancements:
  - Online REORG no longer renames data sets, which greatly reduces the time that data is unavailable during the SWITCH phase.
  - Additional parallel processing improves the elapsed time of the BUILD2 phase of REORG SHRLEVEL(CHANGE) or SHRLEVEL(REFERENCE).
- More concurrency with online LOAD RESUME is achieved by letting you give users read and write access to the data during LOAD processing so that you can load data concurrently with user transactions.
- More efficient processing for SQL queries:
  - More transformations of subqueries into a join for some UPDATE and DELETE statements
  - Fewer sort operations for queries that have an ORDER BY clause and WHERE clauses with predicates of the form COL=*constant*
  - More parallelism for IN-list index access, which can improve performance for queries involving IN-list index access
- The ability to change system parameters without stopping DB2 supports online transaction processing and e-business without interruption.
- Improved availability of user objects that are associated with failed or canceled transactions:
  - You can cancel a thread without performing rollback processing.
  - Some restrictions imposed by the restart function have been removed.
  - A NOBACKOUT option has been added to the CANCEL THREAD command.
- Improved availability of the DB2 subsystem when a log-read failure occurs: DB2 now provides a timely warning about failed log-read requests and the ability to retry the log read so that you can take corrective action and avoid a DB2 outage.
- Improved availability in the data sharing environment:
  - Group attachment enhancements let DB2 applications generically attach to a DB2 data sharing group.
  - A new LIGHT option of the START DB2 command lets you restart a DB2 data sharing member with a minimal storage footprint, and then terminate normally after DB2 frees the retained locks that it can.
  - You can let changes in structure size persist when you rebuild or reallocate a structure.
- Additional data sharing enhancements include:
  - Notification of incomplete units of recovery
  - Use of a new OS/390 and z/OS function to improve failure recovery of group buffer pools
- An additional enhancement for e-business provides improved performance with preformatting for INSERT operations.

## Easier development and integration of e-business applications

Version 7 provides the following enhancements, which let you more easily develop and integrate applications that access data from various DB2 operating systems and distributed environments:

- DB2 XML Extender for OS/390 and z/OS, a new member of the DB2 Extender family, lets you store, retrieve, and search XML documents in a DB2 database.

- Improved support for UNION and UNION ALL operators in a view definition, a nested table expression, or a subquery predicate, improves DB2 family compatibility and is consistent with SQL99 standards.
- More flexibility with SQL gives you greater compatibility with DB2 on other operating systems:
  - Scrollable cursors let you move forward, backward, or randomly through a result table or a result set. You can use scrollable cursors in any DB2 applications that do not use DB2 private protocol access.
  - A search condition in the WHERE clause can include a subquery in which the base object of both the subquery and the searched UPDATE or DELETE statement are the same.
  - A new SQL clause, FETCH FIRST *n* ROWS, improves performance of applications in a distributed environment.
  - Fast implicit close in which the DB2 server, during a distributed query, automatically closes the cursor when the application attempts to fetch beyond the last row.
  - Support for options USER and USING in a new authorization clause for CONNECT statements lets you easily port applications that are developed on the workstation to DB2 for OS/390 and z/OS. These options also let applications that run under WebSphere to reuse DB2 connections for different users and to enable DB2 for OS/390 and z/OS to check passwords.
  - For positioned updates, you can specify the FOR UPDATE clause of the cursor SELECT statement without a list of columns. As a result, all updatable columns of the table or view that is identified in the first FROM clause of the fullselect are included.
  - A new option of the SELECT statement, ORDER BY *expression*, lets you specify operators as the sort key for the result table of the SELECT statement.
  - New datetime ISO functions return the day of the week with Monday as day 1 and every week with seven days.
- Enhancements to Open Database Connectivity (ODBC) provide partial ODBC 3.0 support, including many new application programming interfaces (APIs), which increase application portability and alignment with industry standards.
- Enhancements to the LOAD utility let you load the output of an SQL SELECT statement directly into a table.
- A new component called Precompiler Services lets compiler writers modify their compilers to invoke Precompiler Services and produce an *SQL statement coprocessor*. An SQL statement coprocessor performs the same functions as the DB2 precompiler, but it performs those functions at compile time. If your compiler has an SQL statement coprocessor, you can eliminate the precompile step in your batch program preparation jobs for COBOL and PL/I programs.
- Support for Unicode-encoded data lets you easily store multilingual data within the same table or on the same DB2 subsystem. The Unicode encoding scheme represents the code points of many different geographies and languages.

## Improved connectivity

Version 7 offers improved connectivity:

- Support for COMMIT and ROLLBACK in stored procedures lets you commit or roll back an entire unit of work, including uncommitted changes that are made from the calling application before the stored procedure call is made.

- Support for Windows Kerberos security lets you more easily manage workstation clients who seek access to data and services from heterogeneous environments.
- Global transaction support for distributed applications lets independent DB2 agents participate in a global transaction that is coordinated by an XA-compliant transaction manager on a workstation or a gateway server (Microsoft Transaction Server or Encina, for example).
- Support for a DB2 Connect Version 7 enhancement lets remote workstation clients quickly determine the amount of time that DB2 takes to process a request (the server elapsed time).
- Additional enhancements include:
  - Support for connection pooling and transaction pooling for IBM DB2 Connect
  - Support for DB2 Call Level Interface (DB2 CLI) bookmarks on DB2 UDB for UNIX, Windows, OS/2

## Features of DB2 for OS/390 and z/OS

Version 7 of DB2 UDB Server for OS/390 and z/OS offers several features that help you integrate, analyze, summarize, and share data across your enterprise:

- DB2 Warehouse Manager feature. The DB2 Warehouse Manager feature brings together the tools to build, manage, govern, and access DB2 for OS/390 and z/OS-based data warehouses. The DB2 Warehouse Manager feature uses proven technologies with new enhancements that are not available in previous releases, including:
  - DB2 Warehouse Center, which includes:
    - DB2 Universal Database Version 7 Release 1 Enterprise Edition
    - Warehouse agents for UNIX, Windows, and OS/390
    - Information Catalog
  - QMF Version 7
  - QMF High Performance Option
  - QMF for Windows
- DB2 Management Clients Package. The elements of the DB2 Management Clients Package are:
  - DB2 Control Center
  - DB2 Stored Procedure Builder
  - DB2 Installer
  - DB2 Visual Explain
  - DB2 Estimator
- Net Search Extender for in-memory text search for e-business applications
- Net.Data for secure Web applications

## Migration considerations

Migration with full fallback protection is available when you have either DB2 for OS/390 Version 5 or Version 6 installed. You should ensure that you are fully operational on DB2 for OS/390 Version 5, or later, before migrating to DB2 for OS/390 and z/OS Version 7.

To learn about all of the migration considerations from Version 5 to Version 7, read the *DB2 Release Planning Guide* for Version 6 and Version 7; to learn about content information, also read appendixes A through F in both books.

# Chapter 2. System planning concepts

This chapter introduces the DB2 for OS/390 and z/OS system and explains the concepts that relate to system and database administration. It consists of the following sections:

- "The structure of DB2" describes the elements you deal with when using DB2.
- "Control and maintenance of DB2" on page 15 briefly describes commands and utility jobs.
- "The DB2 environment" on page 18 describes the main DB2 components and explains how DB2 operates with certain related IBM products.

Each section concludes with a list of citations to more detailed information about the topics that the section introduces.

If you are new DB2 for OS/390 and z/OS, begin with *An Introduction to DB2 for OS/390* for extensive conceptual information.

General information about DB2 for OS/390 and z/OS is available from the DB2 for OS/390 and z/OS World Wide Web page:

```
http://www.software.ibm.com/data/db2/os390/
```

## The structure of DB2

The elements that DB2 manages can be divided into two broad categories:

- Data structures, which are accessed under the user's direction and by which the user's data (and some system data) is organized.
- System structures, which are controlled and accessed by DB2.

### Data structures

DB2 data structures described in this section include:

"Databases" on page 9
"Storage groups" on page 9
"Table spaces" on page 9
"Tables" on page 10
"Indexes" on page 10
"Views" on page 11

The brief descriptions here show how the structures fit into an overall view of DB2.

Figure 1 on page 8 shows how some DB2 structures contain others. To some extent, the notion of "containment" provides a hierarchy of structures. This section introduces those structures from the most to the least inclusive.

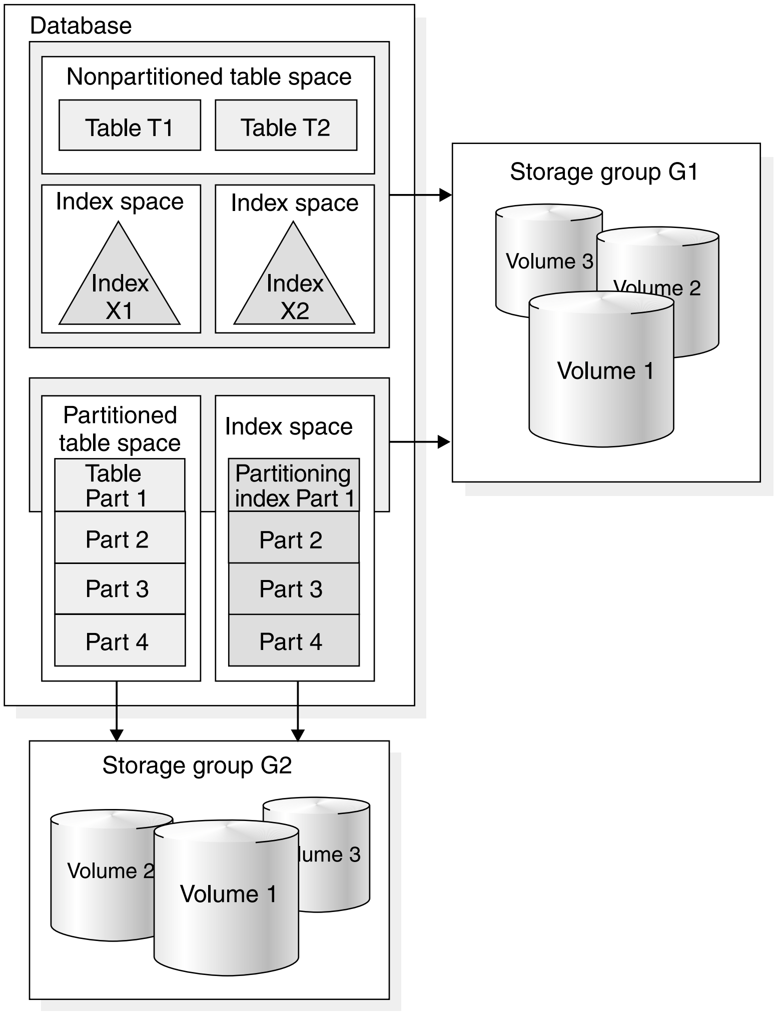


*Figure 1. A hierarchy of DB2 structures*

The DB2 objects that Figure 1 introduces are:

**Databases**
A set of DB2 structures that include a collection of tables, their associated indexes, and the table spaces in which they reside.

**Storage groups**
A set of volumes on disks that hold the data sets in which tables and indexes are actually stored.

**Table spaces**
A set of volumes on disks that hold the data sets in which tables and indexes are actually stored.

**Tables**
All data in a DB2 database is presented in *tables*—collections of rows all having the same columns. A table that holds persistent user data is a *base table*. A table that stores data temporarily is a global temporary table.

**Indexes**
An *index* is an ordered set of pointers to the data in a DB2 table. The index is stored separately from the table.

**Views** A *view* is an alternate way of representing data that exists in one or more tables. A view can include all or some of the columns from one or more base tables.

## Databases

A single database can contain all the data associated with one application or with a group of related applications. Collecting that data into one database allows you to start or stop access to all the data in one operation and grant authorization for access to all the data as a single unit. Assuming that you are authorized to do so, you can access data stored in different databases.

If you create a table space or a table and do not specify a database, the table or table space is created in the default database, DSNDB04. DSNDB04 is defined for you at installation time. All users have the authority to create table spaces or tables in database DSNDB04. The system administrator can revoke those privileges and grant them only to particular users as necessary.

When you migrate to Version 7, DB2 adopts the default database and default storage group you used in Version 6. You have the same authority for Version 7 as you did in Version 6.

## Storage groups

The description of a storage group names the group and identifies its volumes and the VSAM (virtual storage access method) catalog that records the data sets. The default storage group, SYSDEFLT, is created when you install DB2.

All volumes of a given storage group must have the same device type. But, as Figure 1 on page 8 suggests, parts of a single database can be stored in different storage groups.

## Table spaces

A table space can consist of a number of VSAM data sets. Data sets are VSAM linear data sets (LDSs). Table spaces are divided into equal-sized units, called *pages*, which are written to or read from disk in one operation. You can specify page sizes for the data; the default page size is 4 KB.

When you create a table space, you can specify the database to which the table space belongs and the storage group it uses. If you do not specify the database and storage group, DB2 assigns the table space to the default database and the default storage group.

You also determine what kind of table spaces is created.

**Partitioned**
Divides the available space into separate units of storage called *partitions*. Each partition contains one data set of one table. You assign the number of partitions (from 1 to 254) and you can assign partitions independently to different storage groups.

**Segmented**
Divides the available space into groups of pages called *segments*. Each segment is the same size. A segment contains rows from only one table.

**Large object (LOB)**
Holds large object data such as graphics, video, or very large text strings. A LOB table space is always associated with the table space that contains the logical LOB column values. The table space that contains the table with the LOB columns is called, in this context, the *base table space*.

**Simple**

Can contain more than one table. The rows of different tables are not kept separate (unlike segmented table spaces).

## Tables

When you create a table in DB2, you define an ordered set of columns.

**Sample tables:** The examples in this book are based on the set of tables described in Appendix A (Volume 2) of *DB2 Administration Guide*. The sample tables are part of the DB2 licensed program and represent data related to the activities of an imaginary computer services company, the Spiffy Computer Services Company. Table 1 shows an example of a DB2 sample table.

*Table 1. Example of a DB2 sample table (Department table)*

| DEPTNO | DEPTNAME | MGRNO | ADMRDEPT |
|---|---|---|---|
| A00 | SPIFFY COMPUTER SERVICE DIV. | 000010 | A00 |
| B01 | PLANNING | 000020 | A00 |
| C01 | INFORMATION CENTER | 000030 | A00 |
| D01 | DEVELOPMENT CENTER |  | A00 |
| E01 | SUPPORT SERVICES | 000050 | A00 |
| D11 | MANUFACTURING SYSTEMS | 000060 | D01 |
| D21 | ADMINISTRATION SYSTEMS | 000070 | D01 |
| E11 | OPERATIONS | 000090 | E01 |
| E21 | SOFTWARE SUPPORT | 000100 | E01 |

The department table contains:

- **Columns:** The ordered set of columns are DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT. All the data in a given column must be of the same data type.
- **Row:** Each row contains data for a single department.
- **Value:** At the intersection of a column and row is a *value*. For example, PLANNING is the value of the DEPTNAME column in the row for department B01.
- **Referential constraints:** You can assign a *primary key* and *foreign keys* to tables. DB2 can automatically enforce the integrity of references from a foreign key to a primary key by guarding against insertions, updates, or deletions that violate the integrity.
  - **Primary key:** A column or set of columns whose values uniquely identify each row, for example, DEPTNO.
  - **Foreign key:** Columns of other tables, whose values must be equal to values of the primary key of the first table (in this case, the department table). In the sample employee table, the column that shows what department an employee works in is a foreign key; its values must be values of the department number column in the department table.

## Indexes

Each index is based on the values of data in one or more columns of a table. After you create an index, DB2 maintains the index, but you can perform necessary maintenance such as reorganizing it or recovering the index.

Indexes take up physical storage in *index spaces*. Each index occupies its own index space.

The main purposes of indexes are:

- To improve performance. Access to data is often faster with an index than without.
- To ensure that a row is unique. For example, a unique index on the employee table ensures that no two employees have the same employee number.

Except for changes in performance, users of the table are unaware that an index is in use. DB2 decides whether to use the index to access the table. There are ways to influence how indexes affect performance when you calculate the storage size of an index and determine what type of index to use. An index can be partitioning, nonpartitioning, or clustered. For example, you can apportion data by last names, maybe using one partition for each letter of the alphabet. Your choice of a partitioning scheme is based on how an application accesses data, how much data you have, and how large you expect the total amount of data to grow.

### Views

Views allow you to shield some table data from end users. A view can be based on other views or on a combination of views and tables.

When you define a view, DB2 stores the definition of the view in the DB2 catalog. However, DB2 does not store any data for the view itself, because the data already exists in the base table or tables.

## System structures

DB2 system structures described in this section include:

“DB2 catalog”
“DB2 directory” on page 12
“Active and archive logs” on page 12
“Bootstrap data set (BSDS)” on page 13
“Buffer pools” on page 13
“Data definition control support database” on page 14
“Resource limit facility database” on page 14
“Work file database” on page 14
“TEMP database” on page 14

In addition, Parallel Sysplex® data sharing uses shared system structures.

### DB2 catalog

The DB2 catalog consists of tables of data about everything defined to the DB2 system, including table spaces, indexes, tables, copies of table spaces and indexes, storage groups, and so forth. The system database DSNDB06 contains the DB2 catalog.

When you create, alter, or drop any structure, DB2 inserts, updates, or deletes rows of the catalog that describe the structure and tell how the structure relates to other structures. For example, SYSIBM.SYSTABLES is one catalog table that records information when a table is created. DB2 inserts a row into SYSIBM.SYSTABLES that includes the table name, its owner, its creator, and the name of its table space and its database.

Because the catalog consists of DB2 tables in a DB2 database, authorized users can use SQL statements to retrieve information from it.

The *communications database* (CDB) is part of the DB2 catalog. The CDB consists of a set of tables that establish conversations with remote database management systems (DBMSs). The distributed data facility (DDF) uses the CDB to send and receive distributed data requests.

## DB2 directory

The DB2 directory contains information that DB2 uses during normal operation. You cannot access the directory using SQL, although much of the same information is contained in the DB2 catalog, for which you can submit queries. The structures in the directory are not described in the DB2 catalog.

The directory consists of a set of DB2 tables stored in five table spaces in system database DSNDB01. Each of the table spaces listed in Table 2 is contained in a VSAM linear data set.

*Table 2. Directory table spaces*

| Table space name | Description |
|---|---|
| SCT02<br>Skeleton cursor (SKCT) | Contains the internal form of SQL statements contained in an application. When you bind a plan, DB2 creates a skeleton cursor table in SCT02. |
| SPT01<br>Skeleton package | Similar to SCT02 except that the skeleton package table is created when you bind a package. |
| SYSLGRNX<br>Log range | Tracks the opening and closing of table spaces, indexes, or partitions. By tracking this information and associating it with relative byte addresses (RBAs) as contained in the DB2 log, DB2 can reduce recovery time by reducing the amount of log that must be scanned for a particular table space, index, or partition. |
| SYSUTILX<br>System utilities | Contains a row for every utility job that is running. The row stays until the utility is finished. If the utility terminates without completing, DB2 uses the information in the row when you restart the utility. |
| DBD01<br>Database descriptor (DBD) | Contains internal information, called *database descriptors* (DBDs), about the databases that exist within DB2.<br>Each database has exactly one corresponding DBD that describes the database, table spaces, tables, table check constraints, indexes, and referential relationships. A DBD also contains other information about accessing tables in the database. DB2 creates and updates DBDs whenever their corresponding databases are created or updated. |

## Active and archive logs

DB2 records all data changes and significant events in a log as they occur. In the case of failure, DB2 uses this data to recover.

DB2 writes each log record to a disk data set called the *active log*. When the active log is full, DB2 copies the contents of the active log to a disk or magnetic tape data set called the *archive log*.

You can choose either single logging or dual logging.

- A single active log contains between 2 and 31 active log data sets.
- With dual logging, the active log has the capacity for 4 to 62 active log data sets, because two identical copies of the log records are kept.

Each active log data set is a single-volume, single-extent VSAM LDS.

## Bootstrap data set (BSDS)

The *bootstrap data set* (BSDS) is a VSAM key-sequenced data set (KSDS) that contains information critical to DB2. Specifically, the BSDS contains:

- An inventory of all active and archive log data sets known to DB2. DB2 uses this information to track the active and archive log data sets. DB2 also uses this information to locate log records to satisfy log read requests during normal DB2 system activity and during restart and recovery processing.
- A wrap-around inventory of all recent DB2 checkpoint activity. DB2 uses this information during restart processing.
- The distributed data facility (DDF) communication record, which contains information necessary to use DB2 as a distributed server or requester.
- Information about buffer pools.

Because the BSDS is essential to recovery in the event of subsystem failure, during installation DB2 automatically creates two copies of the BSDS and, if space permits, places them on separate volumes.

## Buffer pools

*Buffer pools*, also known as *virtual buffer pools*, are areas of virtual storage in which DB2 temporarily stores pages of table spaces or indexes. When an application program accesses a row of a table, DB2 retrieves the page containing that row and places the page in a buffer. If the needed data is already in a buffer, the application program does not have to wait for it to be retrieved from disk, significantly reducing the cost of retrieving the page.

Buffer pools require monitoring and tuning. The size of buffer pools is critical to the performance characteristics of an application or group of applications that access data in those buffer pools.

When you use Parallel Sysplex data sharing, buffer pools map to structures called *group buffer pools*. These structures reside in a special PR/SM™ LPAR logical partition called a *coupling facility*, which enables several DB2s to share information and control the coherency of data.

There are several options for where buffer pools reside:

- Strictly within DB2's DBM1 primary address space. This option offers the best performance, but limits the amount of space to 1.6 GB.
- Partly within the DBM1 address space, but using extended storage (ESO hiperspace) for infrequently updated data ("clean" data). Using extended storage expands the storage capacity to 1.6 GB of primary and 8 GB of extended storage. DB2 must move the data back into the DBM1 address space to address it.
- Solely within an MVS *data space*. Data spaces greatly expand capacity and are provided to position DB2 for future S/390® processor enhancements that will provide large real memory.

If storage constraints in DB2's DBM1 address space are likely to be a problem for your site, consider the hiperspace and data space options.

***Buffer pools in data spaces:*** A buffer pool in a data space can support up to 8 million buffers. For a 32 KB buffer pool, that is 256 gigabytes of virtual storage. Because of these very large sizes, a buffer pool can span multiple data spaces, although a single data space never has more than one buffer pool in it.

***Buffer pools in hiperspace:*** Mutually exclusive from data spaces is the option to store clean data in extended storage, called *hiperpools*.

The second level of storage, the *hiperpool*, is an extension to the virtual buffer pool. Virtual buffer pools hold the most frequently accessed data. Clean data in virtual buffer pools that is not accessed frequently can be moved to its corresponding hiperpool—only one hiperpool can exist for each virtual buffer pool.

Hiperpools can span up to four *hiperspaces*, 2 GB expanded storage areas. Using hiperspaces and hiperpools improves performance because you can cache up to 8 GB to help avoid I/O operations.

### Data definition control support database

The *data definition control support database* is automatically created during installation. This database is a user-maintained collection of tables used by data definition control support to restrict the submission of specific DB2 DDL (data definition language) statements to selected application identifiers (plans or collections of packages). After this database is created, you must populate the tables to use of this facility. The system name for this database is DSNRGFDB.

### Resource limit facility database

The *resource limit facility database* (DSNRLST) is a facility that lets you control the amount of processor resources used by dynamic SELECT statements. For example, you might choose to disable bind operations during critical times of day to avoid contention with the DB2 catalog.

You can establish a single limit for all users, different limits for individual users, or both. You can choose to have these limits applied before the statement is executed (this is called *predictive* governing), or while a statement is running (sometimes called *reactive* governing). You can even use both modes of governing. You define these limits in one or more resource limit specification tables (RLST).

### Work file database

The *work file database* is used as storage for processing SQL statements that require working space, such as that required for a sort. DB2 creates a work file database for you at installation time, and you can create additional work file table spaces at any time using CREATE TABLESPACE statements.

In a non-data-sharing environment, the work file database is called DSNDB07. In a data sharing environment, each DB2 member in the data sharing group has its own work file database.

### TEMP database

The TEMP database is for declared temporary tables only. DB2 stores all declared temporary tables in this database. You can create one TEMP database for each DB2 subsystem or data sharing member.

## More information about data structures

Table 3 on page 15 lists additional information sources about topics that this section introduces.

*Table 3. More information about DB2 structures*

| For more information about... | See... |
|---|---|
| Basic concepts for designing data structures, including:<br>• Table spaces<br>• Tables, views<br>• Columns<br>• Indexes | *An Introduction to DB2 for OS/390* |
| **Data structures** | |
| Data structures, defining | • *An Introduction to DB2 for OS/390*<br>• “Chapter 5. Implementing your design” on page 41 |
| Table space size limits | Appendix A of *DB2 SQL Reference* |
| Table columns, data types | Volume 1 of *DB2 SQL Reference* |
| Referential integrity | Volume 1 of *DB2 Application Programming and SQL Guide* |
| **System structures** | |
| Shared system structures | *DB2 Data Sharing: Planning and Administration* |
| Catalog tables | Appendix D of *DB2 SQL Reference* |
| Catalog, data set naming conventions | *DB2 Installation Guide* |
| CDB | *DB2 Installation Guide* |
| Directory, data set naming conventions | *DB2 Installation Guide* |
| Logs | “Chapter 18. Managing the log and the bootstrap data set” on page 331 |
| BSDS usage, functions | “Managing the bootstrap data set (BSDS)” on page 341 |
| Buffer pools, tuning | • “Chapter 27. Tuning DB2 buffer, EDM, RID, and sort pools” on page 549<br>• *DB2 Command Reference* |
| Group buffer pools | *DB2 Data Sharing: Planning and Administration* |
| Data definition control support database | “Chapter 11. Controlling access through a closed application” on page 157 |
| RLST | “Resource limit facility (governor)” on page 581 |
| Work file and TEMP database, defining | Volume 2 of *DB2 SQL Reference* |

## Control and maintenance of DB2

You use commands and utilities to perform the tasks required to control and maintain DB2:

- Commands can be entered at a terminal, an MVS console, or through an APF-authorized program or application that uses the instrumentation facility interface (IFI)
- Utility jobs run as standard MVS batch jobs

## Commands

The commands are divided into the following categories:
- DSN command and subcommands
- DB2 commands
- IMS commands
- CICS attachment facility commands
- MVS IRLM commands
- TSO CLIST commands

To enter a DB2 command from an authorized MVS console, you use a subsystem command prefix (composed of 1 to 8 characters) at the beginning of the command. The default subsystem command prefix is *-DSN1*, which you can change when you install or migrate DB2.

**Example:** The following command starts the DB2 subsystem that is associated with the command prefix -DSN1:

```
-DSN1 START DB2
```

## Utilities

You use utilities to perform many of the tasks required to maintain DB2 data. Those tasks include loading a table, copying a table space, or recovering a database to a previous point in time.

The utilities run as batch jobs under MVS. *DB2 interactive* (DB2I) provides a simple way to prepare the job control language (JCL) for those jobs and to perform many other operations by entering values on panels. DB2I runs under TSO using ISPF services. A utility control statement tells a particular utility what task to perform.

## High availability

It is not necessary to start or stop DB2 often. DB2 continually adds function to improve availability, especially in the following areas:
- "Daily operations and tuning"
- "Backup and recovery"
- "Restart" on page 17

### Daily operations and tuning

Some of the high availability features related to normal DB2 operations include:

- You can bind application plans and packages online. Packages let you change and rebind smaller units. Using package *versions* permits binding while the applications continue to run.
- You can define and change databases and authorizations online.
- You can change buffer pool sizes online.
- You can use utilities to reorganize indexes, table spaces, or partitions of indexes or table spaces.
- DB2's data sharing function lets several DB2 subsystems process applications on shared data. While the different subsystems share data, they appear as a single DB2 to end users. Applications can be rerouted to avoid outages if one of the subsystems must be taken down for maintenance.

### Backup and recovery

Unplanned outages are difficult to avoid entirely. However, a good backup strategy can reduce the elapsed time of an unplanned outage. To reduce the probability and duration of unplanned outages, you should periodically back up and reorganize your data.

A lot of factors affect the availability of the databases. Here are some key points to be aware of:

- You should limit your use of, and understand the options of, utilities such as COPY and REORG.
  – You can recover online such structures as table spaces, partitions, data sets, a range of pages, a single page, and indexes.
  – You can recover table spaces and indexes at the same time to reduce recovery time.
  – With some options on the COPY utility, you can read and update a table space while copying it.
- I/O errors have the following affects:
  – I/O errors on a range of data do not affect availability to the rest of the data.
  – If an I/O error occurs when DB2 is writing to the log, DB2 continues to operate.
  – If an I/O error is on the active log, DB2 moves to the next data set. If the error is on the archive log, DB2 dynamically allocates another data set.
- Documented disaster recovery methods are crucial in the case of disasters that might cause a complete shutdown of your local DB2 system.
- If DB2 is forced to a single mode of operations for the bootstrap data set or logs, you can usually restore dual operation while DB2 continues to run.

### Restart

A key to the perception of high availability is getting the DB2 subsystem back up and running quickly after an unplanned outage.

- Some restart processing can occur concurrently with new work. Also, you can choose to postpone some processing.
- During a restart, DB2 applies data changes from its log that was not written at the time of failure. Some of this process can be run in parallel.
- You can register DB2 to the Automatic Restart Manager of OS/390. This facility automatically restarts DB2 should it go down as a result of a failure.

## More information about control and maintenance of DB2

Table 4 lists additional information sources about topics that this section introduces.

*Table 4. More information about DB2 control and maintenance*

| For more information about... | See... |
|---|---|
| Commands | • “Part 4. Operation and recovery” on page 241<br>• *DB2 Command Reference* |
| Utilities | *DB2 Utility Guide and Reference* |
| Data sharing | *DB2 Data Sharing: Planning and Administration* |
| Recovery | • “Recovering page sets and data sets” on page 393<br>• *DB2 Utility Guide and Reference* |

# The DB2 environment

This section provides an overview of DB2 components and environments that work together in the OS/390 and z/OS environment. DB2 operates as a formal subsystem of OS/390. DB2 utilities run in the batch environment, and applications that access DB2 resources can run in the batch, TSO, IMS, or CICS environments. IBM provides attachment facilities to connect DB2 to each of these environments.

## Address spaces

DB2 uses several different address spaces for the following purposes:

**Database services**
*ssnm*DBM1 manipulates most of the structures in user-created databases.

**System services**
*ssnm*MSR performs a variety of system-related functions.

**Distributed data facility**
*ssnm*DIST provides support for remote requests.

**IRLM (Internal resource lock manager)**
IRLMPROC controls DB2 locking.

**DB2-established**
*ssnm*SPAS, for stored procedures, provides an isolated execution environment for user-written SQL programs at a DB2 server.

**WLM-established**
Zero to many address spaces for stored procedures and user-defined functions. WLM-established address spaces are handled in order of priority and isolated from other stored procedures or user-defined functions running in other address spaces

**User address spaces**
At least one, possibly several, of the following types of user address spaces:

- TSO
- Batch
- CICS
- IMS dependent region
- IMS control region

## DB2’s lock manager

DB2’s internal resource lock manager (IRLM) is both a separate subsystem and an integral component of DB2. IRLM is shipped with DB2, and each DB2 subsystem must have its own instance of IRLM.

**Recommendation:** Always run with the latest level of IRLM.

You cannot share IRLM between DB2s or between DB2 and IMS. (IRLM is also shipped with IMS.) If you are running a DB2 data sharing group, there is a corresponding IRLM group.

### What IRLM does

IRLM works with DB2 to serialize access to your data. DB2 requests locks from IRLM to ensure data integrity when applications, utilities, commands, and so forth, are all attempting to access the same data.

### Administering IRLM

IRLM requires some control and monitoring. The external interfaces to the IRLM include:

- Installation

  Install IRLM when you install DB2. Consider that locks take up storage, and adequate storage for IRLM is crucial to the performance of your system.

  Another important performance item is to make the priority of the IRLM address space above all the DB2 address spaces.

- Commands

  Some MVS commands specifically for IRLM let you modify parameters, display information about the status of the IRLM and its storage use, and start and stop IRLM.

- Tracing

  DB2's trace facility gives you the ability to trace lock interactions.

  IRLM uses the MVS component trace services for its diagnostic traces. You normally use these under the direction of IBM Service.

## DB2's attachment facilities

This section describes the attachment facilities that you can use in the OS/390 environment to begin a DB2 session. You can also begin DB2 sessions from other environments on clients such as Windows® or UNIX® by using interfaces that include ODBC, JDBC™, and SQLJ.

An *attachment facility* provides the interface between DB2 and another environment. Figure 2 shows the OS/390 attachment facilities with interfaces to DB2.

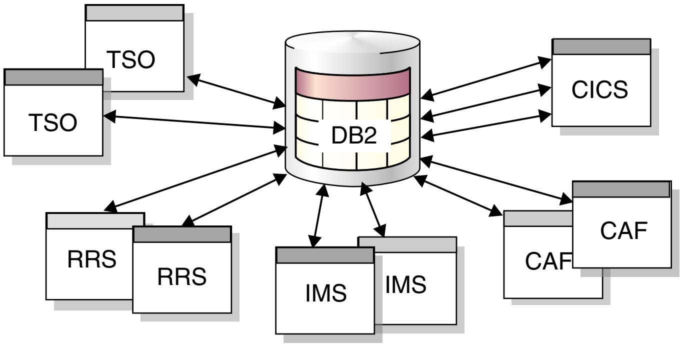


*Figure 2. Attaching to DB2*

The OS/390 environments include:

- CICS (Customer Information Control System)
- IMS (Information Management System)
- TSO (Time Sharing Option)
- Batch

The OS/390 attachment facilities include:

- CICS
- IMS
- TSO
- CAF (call attachment facility)
- RRS (Resource Recovery Services)

In the TSO and batch environments, you can use the TSO, CAF, and RRS attachment facilities to access DB2.

## CICS

The *Customer Information Control System* (CICS) attachment facility provided with the CICS transaction server lets you access DB2 from CICS. After you start DB2, you can operate DB2 from a CICS terminal. You can start and stop CICS and DB2 independently, and you can establish or terminate the connection between them at any time. You also have the option of allowing CICS to connect to DB2 automatically.

The CICS attachment facility also provides CICS applications with access to DB2 data while operating in the CICS environment. CICS applications, therefore, can access both DB2 data and CICS data. In case of system failure, CICS coordinates recovery of both DB2 and CICS data.

***CICS operations:*** The CICS attachment facility uses standard CICS command-level services where needed.

**Examples:**

```
EXEC CICS WAIT
EXEC CICS ABEND
```

A portion of the CICS attachment facility executes under the control of the transaction issuing the SQL requests. Therefore these calls for CICS services appear to be issued by the application transaction.

With proper planning, you can include DB2 in a CICS XRF recovery scenario.

***Application programming with CICS:*** Programmers writing CICS command-level programs can use the same data communication coding techniques to write the data communication portions of application programs that access DB2 data. Only the database portion of the programming changes. For the database portions, programmers use SQL statements to retrieve or modify data in DB2 tables.

To a CICS terminal user, application programs that access both CICS and DB2 data appear identical to application programs that access only CICS data.

DB2 supports this cross-product programming by coordinating recovery resources with those of CICS. CICS applications can therefore access CICS-controlled resources as well as DB2 databases.

Function shipping of SQL requests is not supported. In a CICS multi-region operation (MRO) environment, each CICS address space can have its own attachment to the DB2 subsystem. A single CICS region can be connected to only one DB2 subsystem at a time.

***System administration and operation with CICS:*** An authorized CICS terminal operator can issue DB2 commands to control and monitor both the attachment facility and DB2 itself. Authorized terminal operators can also start and stop DB2 databases.

Even though you perform DB2 functions through CICS, you need to have the TSO attachment facility and ISPF to take advantage of the online functions supplied with DB2 to install and customize your system. You also need the TSO attachment to bind application plans and packages.

## IMS

The *Information Management System* (IMS) attachment facility allows you to access DB2 from IMS. The IMS attachment facility receives and interprets requests for access to DB2 databases using exits provided by IMS subsystems. Usually, IMS connects to DB2 automatically with no operator intervention.

In addition to Data Language I (DL/I) and Fast Path calls, IMS applications can make calls to DB2 using embedded SQL statements. In case of system failure, IMS coordinates recovery of both DB2 and IMS data.

With proper planning, you can include DB2 in an IMS XRF recovery scenario.

***Application programming with IMS:*** With the IMS attachment facility, DB2 provides database services for IMS dependent regions. DL/I batch support allows users to access both IMS data (DL/I) and DB2 data in the IMS batch environment, which includes:

- Access to DB2 and DL/I data from application programs.
- Coordinated recovery through a two-phase commit process.
- Use of the IMS extended restart (XRST) and symbolic checkpoint (CHKP) calls by application programs to coordinate recovery with IMS, DB2, and generalized sequential access method (GSAM) files.

IMS programmers writing the data communication portion of application programs do not need to alter their coding technique to write the data communication portion when accessing DB2; only the database portions of the application programs change. For the database portions, programmers code SQL statements to retrieve or modify data in DB2 tables.

To an IMS terminal user, IMS application programs that access DB2 appear identical to IMS.

DB2 supports this cross-product programming by coordinating database recovery services with those of IMS. Any IMS program uses the same synchronization and rollback calls in application programs that access DB2 data as they use in IMS DB/DC application programs that access DL/I data.

Another aid for cross-product programming is the DataPropagator NonRelational (DPropNR) licensed program. DPropNR allows automatic updates to DB2 tables when corresponding information in an IMS database is updated, and it allows automatic updates to an IMS database when a DB2 table is updated.

***System administration and operation with IMS:*** An authorized IMS terminal operator can issue DB2 commands to control and monitor DB2. The terminal operator can also start and stop DB2 databases.

Even though you perform DB2 functions through IMS, you need the TSO attachment facility and ISPF to take advantage of the online functions supplied with DB2 to install and customize your system. You also need the TSO attachment facility to bind application plans and packages.

## TSO

The *Time Sharing Option* (TSO) attachment facility is required for binding application plans and packages and for executing several online functions that are provided with DB2.

Using the TSO attachment facility, you can access DB2 by running in either foreground or batch. You gain foreground access through a TSO terminal; you gain batch access by invoking the TSO terminal monitor program (TMP) from an MVS batch job.

The following two command processors are available:

- DSN command processor — Runs as a TSO command processor and uses the TSO attachment facility.
- DB2 Interactive (DB2I) — Consists of Interactive System Productivity Facility (ISPF) panels. ISPF has an interactive connection to DB2, which invokes the DSN command processor. Using DB2I panels, you can perform most DB2 tasks interactively, such as running SQL statements, commands, and utilities.

Whether you access DB2 in foreground or batch, attaching through the TSO attachment facility and the DSN command processor makes access easier. DB2 subcommands that execute under DSN are subject to the command size limitations as defined by TSO. TSO allows authorized DB2 users or jobs to create, modify, and maintain databases and application programs. You invoke the DSN processor from the foreground by issuing a command at a TSO terminal. From batch, first invoke TMP from within an MVS batch job, and then pass commands to TMP in the SYSTSIN data set.

After DSN is running, you can issue DB2 commands or DSN subcommands. You cannot issue a -START DB2 command from within DSN. If DB2 is not running, DSN cannot establish a connection to it; a connection is required so that DSN can transfer commands to DB2 for processing.

### CAF

Most TSO applications must use the TSO attachment facility, which invokes the DSN command processor. Together, DSN and TSO provide services such as automatic connection to DB2, attention key support, and translation of return codes into error messages. However, when using DSN services, your application must run under the control of DSN.

The *call attachment facility* (CAF) provides an alternative connection for TSO and batch applications needing tight control over the session environment. Applications using CAF can *explicitly* control the state of their connections to DB2 by using connection functions that CAF supplies.

### RRS

OS/390 Resource Recovery Services (RRS) is a feature of OS/390 that coordinates two-phase commit processing of recoverable resources in an MVS system. DB2 supports use of these services for DB2 applications that use the RRS attachment facility provided with DB2. Use the RRS attachment to access resources such as SQL tables, DL/I databases, MQSeries® messages, and recoverable VSAM files within a single transaction scope.

The RRS attachment is required for stored procedures that run in a WLM-established address space.

## DB2 and distributed data

In a distributed data environment, DB2 applications can access data at many different DB2 sites and at remote relational database systems.

**Example:** Assume a company needs to satisfy customer requests at hundreds of locations and the company representatives who answer those requests work at locations that span a wide geographic area. You can document requests on workstations that have DB2 Connect® Personal Edition. This information is uploaded to DB2 for OS/390 and z/OS. The representatives can then use Java applications to access the customer request information in DB2 from their local offices.

The company’s distributed environment relies on the distributed data facility (DDF), which is part of DB2 for OS/390 and z/OS. DB2 applications can use DDF to access data at other DB2 sites and at remote relational database systems that support *Distributed Relational Database Architecture* (DRDA). *DRDA* is a standard for distributed connectivity. All IBM DB2 servers support this DRDA standard.

DDF also enables applications that run in a remote environment that supports DRDA. These applications can use DDF to access data in DB2 servers. Examples of application requesters include IBM DB2 Connect and other DRDA-compliant client products.

With DDF, you can have up to 150 000 distributed *threads* connect to a DB2 server at the same time. A thread is a DB2 structure that describes an application's connection and traces its progress.

Use *stored procedures* to reduce processor and elapsed time costs of distributed access. A stored procedure is user-written SQL program that a requester can invoke at the server. By encapsulating the SQL, many fewer messages flow across the wire.

Local DB2 applications can use stored procedures as well to take advantage of the ability to encapsulate SQL that is shared among different applications.

The decision to access distributed data has implications for many DB2 activities: application programming, data recovery, authorization, and so on.

## DB2 and OS/390 and z/OS

z/OS is the next generation of the OS/390 operating system. z/OS and the IBM @server zSeries 900 server offer architecture that provides qualities of service that are critical for e-business. The z/OS operating system is based on 64-bit z/Architecture. The operating system is highly secure, scalable, and high performing. With these characteristics, z/OS provides a strong base for Internet and Java-enabled applications and a comprehensive and diverse environment for running your applications.

OS/390 is the operating system software for the IBM System/390® family of enterprise servers. At the core of OS/390 is the base control program, MVS.

As a formal subsystem of OS/390, DB2 uses:

- Availability and scalability features that include System/390 Parallel Sysplex cluster technology that enables multiple processors to perform work.
- VTAM and TCP/IP for distributed data facility
- Reliability features that include protection from unplanned outages and recovery routines
- Serviceability features that include:
  SYS1.LOGREC

SYS1.DUMP

- Synchronous cross-memory services for address space switching
- System Management Facilities (SMF) for statistics, accounting information, and performance data

## DB2 and the Parallel Sysplex

The Parallel Sysplex is a key example of the synergy of DB2 and System/390. DB2 takes advantage of the System/390 Parallel Sysplex, with its superior processing capabilities. By allowing two or more processors to share the same data, you can maximize performance while minimizing cost; improve system availability and concurrency; expand system capacity; and configure your system environment more flexibly. With data sharing, applications running on more than one DB2 subsystem can read from and write to the same set of data concurrently.

Sharing DB2s must belong to a DB2 data sharing *group*. A data sharing group is a collection of one or more DB2 subsystems accessing shared DB2 data. Each DB2 subsystem belonging to a particular data sharing group is a *member* of that group. All members of a group use the same shared DB2 catalog and directory.

With data sharing, you can grow your system incrementally by adding additional central processor complexes and DB2s to the data sharing group. You don't have to move part of the workload onto another system, alleviating the need to manage copies of the data or to use distributed processing to access the data.

You can configure your environment flexibly. For example, you can tailor each OS/390 image to meet the requirements for the user set on that image. For processing that occurs during peak workload periods, you can bring up a dormant DB2 to help process the work.

## DB2 and the SecureWay Security Server for OS/390

You can use the Resource Access Control Facility (RACF) component of the SecureWay Security Server for OS/390, or an equivalent product, to control access to your OS/390 system. When users begin sessions with TSO, IMS, or CICS, their identities are checked to prevent unauthorized access to the system.

**Recommendation:** Use the Security Server check the security of DB2 users and to protect DB2 resources. The Security Server provides effective protection for DB2 data by permitting only DB2-mediated access to DB2 data sets.

Much authorization to DB2 objects can be controlled directly from the Security Server. An *exit routine* (a program that runs as an extension of DB2) that is shipped with the OS/390 Security Server lets you centralize access control.

## DB2 and DFSMS

The DFSMSdfp™ *storage management subsystem* (SMS) can be used to manage DB2 disk data sets. The purpose of SMS is to automate as much as possible the management of physical storage by centralizing control, automating tasks, and providing interactive controls for system administrators. SMS can reduce users' needs to be concerned about physical details of performance, space, and device management.

Consult with your site's storage administrator about using SMS for DB2 private data, image copies, and archive logs. For data that is especially performance-sensitive, there might need to be more manual control over data set placement.

Table spaces or indexes with data sets larger than 4 gigabytes require SMS-managed data sets.

Extended partitioned data sets (PDSE), a feature of DFSMSdfp, are useful for managing stored procedures that run in a stored procedures address space. PDSE enables extent information for the load libraries to be dynamically updated, reducing the need to start and stop the stored procedures address space.

## More information about the OS/390 environment

Table 5 lists additional information sources about topics that this section introduces.

*Table 5. More information about the OS/390 environment*

| For more information about... | See... |
|---|---|
| z/OS | www.ibm.com/servers/eserver/zseries/zos/ |
| OS/390 | www.s390.ibm.com/os390/ |
| IRLM installation | *DB2 Installation Guide* |
| IRLM address spaces | "Setting address space priority" on page 614 |
| IRLM commands | *DB2 Command Reference* |
| IRLM lock tracing | "Using the statistics and accounting traces to monitor locking" on page 701 |
| Exit routines | Appendix B (Volume 2) of *DB2 Administration Guide* |
| Security methods | "Part 3. Security and auditing" on page 93 |
| PDSE data sets | *MVS/DFP: Using Data Sets* |
| SMS | "Using SMS to archive log data sets" on page 337 |
| DFSMShsm | "Managing your DB2 data sets with DFSMShsm™" on page 37 |
| Attachment facilities, programming | Volume 2 of *DB2 Application Programming and SQL Guide* |
| CICS XRF | • "Extended recovery facility (XRF) toleration" on page 374<br>• *CICS for MVS/ESA Operations and Utilities Guide* |
| CICS connections | "Chapter 17. Monitoring and controlling DB2 and its connections" on page 267 |
| CICS administration | *DB2 Installation Guide* |
| IMS XRF | • "Extended recovery facility (XRF) toleration" on page 374<br>• *IMS Administration Guide: System* |
| DL/I batch | Volume 2 of *DB2 Application Programming and SQL Guide* |
| DataPropagator NonRelational | *DataPropagator NonRelational MVS/ESA Administration Guide* |

*Table 5. More information about the OS/390 environment (continued)*

| For more information about... | See... |
|---|---|
| ISPF | Volume 2 of *DB2 Application Programming and SQL Guide* |
| Distributed data | Volume 1 of *DB2 Application Programming and SQL Guide* |
| Parallel Sysplex data sharing | *DB2 Data Sharing: Planning and Administration* |

# Part 2. Designing a database: advanced topics

# Chapter 3. Introduction to designing a database: advanced topics

The scope of “Part 2. Designing a database: advanced topics” on page 27, formerly entitled,″Designing a database,″ has been changed in Version 7. In previous versions, ″Designing a database″ provided a range of information, from basic to advanced, about designing a database. “Part 2. Designing a database: advanced topics” on page 27 now presents only the advanced topics. The newest member of the DB2 for OS/390 and z/OS library, *An Introduction to DB2 for OS/390*, covers basic information about designing and implementing a database.

Table 6 shows where you can find more information about topics related to designing a database.

*Table 6. More information about designing a database*

| For more information about... | See... |
|---|---|
| Basic database design concepts for DB2 Universal Database for OS/390 and z/OS, including:<br>• Designing tables and views<br>• Designing columns<br>• Designing indexes<br>• Designing table spaces | *An Introduction to DB2 for OS/390* |
| Maintaining data integrity, including:<br>• Maintaining referential constraints<br>• Defining table check constraints<br>• Planning to use triggers | Part 2 of *DB2 Application Programming and SQL Guide* |
| Maintaining data integrity, including implications for the following SQL statements: INSERT, UPDATE, DELETE, and DROP | Chapter 5 of *DB2 SQL Reference* |
| Maintaining data integrity, including implications for the following utilities: COPY, QUIESCE, RECOVER, and REPORT | Part 2 of *DB2 Utility Guide and Reference* |
| Detailed information on partitioning and nonpartitioning indexes | Chapter 5 of *DB2 SQL Reference* |
| Compressing data in a table space or a partition | Part 5 (Volume 2) of *DB2 Administration Guide* |

# Chapter 4. Creating storage groups and managing DB2 data sets

This chapter provides information on how to create storage groups and manage your DB2 data sets:

DB2 manages the auxiliary storage requirements of a DB2 database by using *DB2 storage groups*. Data sets in these DB2 storage groups are *DB2-managed data sets*. These DB2 storage groups are not the same as storage groups defined by DFSMS’s storage management subsystem (DFSMS). A DB2 storage group is a named set of disk volumes, in which DB2:

- Allocates storage for table spaces and indexes
- Defines the necessary VSAM data sets
- Extends and deletes the VSAM data sets
- Alters VSAM data sets

## Creating DB2 storage groups

A name for DB2 storage groups and databases is an unqualified identifier of up to eight characters. A DB2 storage group name must not be the same as the name of any other storage group in the DB2 catalog, and a DB2 database name must not be the same as the name of any other DB2 database. The following examples are used in the sample application:

| **Object** | **Name** |
|---|---|
| **DB2 storage group** | DSN8G710 |
| **Database** | DSN8D71A |

See the *DB2 SQL Reference* for more information about naming conventions.

To create a DB2 storage group, use the SQL statement CREATE STOGROUP. For detailed information on CREATE STOGROUP, see Chapter 5 of *DB2 SQL Reference*.

When you create table spaces and indexes, you name the storage group from which you want space to be allocated. Try to assign frequently accessed objects (indexes, for example) to fast devices, and assign seldom-used tables to slower devices. This approach to choosing storage groups improves performance.

Here are some of the things that DB2 does for you in managing your auxiliary storage requirements:

- When a table space is created, DB2 defines the necessary VSAM data sets using VSAM access method services. After the data sets are created, you can process them with access method service commands that support VSAM control-interval (CI) processing (for example, IMPORT and EXPORT).

  **Exception:** You can defer the allocation of data sets for table spaces and index spaces by specifying the DEFINE NO clause on the associated statement (CREATE TABLESPACE and CREATE INDEX), which also must specify the USING STOGROUP clause. For more information about deferring data set allocation, see either “Deferring allocation of data sets for table spaces” on page 36 or Chapter 5 of *DB2 SQL Reference*.

- When a table space is dropped, DB2 automatically deletes the associated data sets.
- When a data set in a segmented or simple table space reaches its maximum size of 2 GB, DB2 might automatically create a new data set. The primary data set allocation is obtained for each new data set.
- When needed, DB2 can extend individual data sets. For more information, see “Extending DB2-managed data sets” on page 39.
- When creating or reorganizing a table space that has associated data sets, DB2 deletes and then redefines them. However, when you run REORG with the REUSE parameter and SHRLEVEL NONE, REORG resets and reuses DB2–managed data sets without deleting and redefining them.
- When you want to move data sets to a new volume, you can alter the volumes list in your storage group. DB2 automatically relocates your data sets during utility operations that build or rebuild a data set (LOAD REPLACE, REORG, REBUILD, and RECOVER). To move your user-defined data sets, you must delete and redefine your data sets.

After you define a storage group, DB2 stores information about it in the DB2 catalog. (This catalog is *not* the same as the integrated catalog facility catalog that describes DB2 VSAM data sets). The catalog table SYSIBM.SYSSTOGROUP has a row for each storage group, and SYSIBM.SYSVOLUMES has a row for each volume. With the proper authorization, you can display the catalog information about DB2 storage groups by using SQL statements. See Appendix D of *DB2 SQL Reference* for more information about using SQL statements to display catalog information about DB2 storage groups.

A *default storage group*, SYSDEFLT, is defined when DB2 is installed. If you are authorized and do not take specific steps to manage your own storage, you can still define tables, indexes, table spaces, and databases; DB2 uses SYSDEFLT to allocate the necessary auxiliary storage. Information about SYSDEFLT, as with any other storage group, is kept in the catalog tables SYSIBM.SYSSTOGROUP and SYSIBM.SYSVOLUMES.

Use storage groups whenever you can, either specifically or by default. However, if you want to maintain closer control over the physical storage of your tables and indexes, you can define and manage your own VSAM data sets using VSAM access method services. See “Managing your own DB2 data sets” on page 33 for more information about managing VSAM data sets.. Yet another possibility is to let SMS manage some or all of your DB2 data sets. See “Managing your DB2 data sets with DFSMShsm™” on page 37 for more information.

When defining DB2 storage groups, use the VOLUMES(’*’) attribute on the CREATE STOGROUP statement to let SMS control the selection of volumes during allocation. See “Managing your DB2 data sets with DFSMShsm™” on page 37 for more information. Otherwise, if you use DB2 to allocate data to specific volumes, you must assign an SMS Storage Class with Guaranteed Space, and you must manage free space for each volume to prevent failures during the initial allocation and extension. Using Guaranteed Space reduces the benefits of SMS allocation, requires more time for space management, and can result in more space shortages. You should only use Guaranteed Space when space needs are relatively small and do not change.

For both user-managed and DB2-managed data sets, you need at least one integrated catalog facility catalog, either user or master, created with the integrated catalog facility. Recommendation: Let SMS manage your DB2 storage groups, you can use asterisks (nonspecific volume IDs) in the VOLUMES clause. You must identify the catalog of the integrated catalog facility (known as the integrated catalog) when you create a storage group or when you create a table space that does not use storage groups.

## Defining index space storage

Generally, the CREATE INDEX statement creates an index space in the same DB2 database that contains the table on which the index is defined. This is true even if you defer building the index.

**Exceptions:**

- If you specify the USING VCAT clause, you create and manage the data sets yourself.
- If you specify the DEFINE NO clause on a CREATE INDEX statement that uses the USING STOGROUP clause, DB2 defers the allocation of the data sets for the index space.

When you use CREATE INDEX, always specify a USING clause. When you specify USING, you declare whether you want DB2-managed or user-managed data sets. For DB2-managed data sets, you specify the primary and secondary space allocation parameters on the CREATE INDEX statement. If you do not specify USING, DB2 assigns the index data sets to the default storage groups using default space attributes. For information about how space allocation can affect the performance of mass inserts, see “Speed up preformatting by allocating in cylinders” on page 540.

You can specify the USING clause to allocate space for the entire index, or, if the index is a partitioning index, you can allocate space for each partition. Information about space allocation for the index is kept in the SYSIBM.SYSINDEXPART table of the DB2 catalog. Other information about the index is in SYSIBM.SYSINDEXES. For more information about determining the space required for an index, see “Calculating the space required for an index” on page 88. For more information about CREATE INDEX clauses, see Chapter 5 of *DB2 SQL Reference*.

## Managing your own DB2 data sets

You might choose to manage your own VSAM data sets for reasons such as these:

- You have a large linear table space on several data sets. If you manage your own data sets, you can better control the placement of individual data sets on the volumes. (Although you can keep a similar type of control by using single-volume DB2 storage groups.)
- You want to prevent deleting a data set within a specified time period, by using the TO and FOR options of the access method services DEFINE and ALTER commands. You can create and manage the data set yourself, or you can create the data set with DB2 and use the ALTER command of access method services to change the TO and FOR options.
- You are concerned about recovering dropped table spaces. Your own data set is not automatically deleted when a table space is dropped, making it easier to reclaim the data if the table space is dropped.

## Managing your data sets using access method services

To manage DB2 auxiliary storage yourself, you use access method services. To define the required data sets, use DEFINE CLUSTER; to add secondary volumes to expanding data sets, use ALTER ADDVOLUMES; and to delete data sets, use DELETE CLUSTER.

You can define a data set for each of these items:
- A simple or segmented table space
- A partition of a partitioned table space
- A nonpartitioning index
- A partition of a partitioning index

Furthermore, as table spaces and index spaces expand, you might need to provide additional data sets. To take advantage of parallel I/O streams when doing certain read-only queries, consider spreading large table spaces over different disk volumes that are attached on separate channel paths. For more information about data set extension, see “Extending DB2-managed data sets” on page 39.

## Requirements for your own data sets

DB2 checks whether you have defined your data sets correctly. If you plan to define and manage VSAM data sets yourself, you must perform these steps:

1. Define the data sets *before* you issue the CREATE TABLESPACE or the CREATE INDEX statement.

   If you create a partitioned table space, you must create a separate data set for each partition, or allocate space for each partition by using the PART option of the NUMPARTS clause.

2. Give each data set a name with this format:

   ```
   catname.DSNDBx.dbname.psname.y0001.Annn
   ```

   **catname**
   Integrated catalog name or alias (up to eight characters). Use the same name or alias here as in the USING VCAT clause of the CREATE TABLESPACE and CREATE INDEX statements.

   **x** C (for VSAM clusters) or D (for VSAM data components).

   **dbname**
   DB2 database name. If the data set is for a table space, *dbname* must be the name given in the CREATE TABLESPACE statement. If the data set is for an index, *dbname* must be the name of the database containing the base table. If you are using the default database, *dbname* must be DSNDB04.

   **psname**
   Table space name or index name. This name must be unique within the database.

   You use this name on the CREATE TABLESPACE or CREATE INDEX statement. (You can use a name longer than eight characters on the CREATE INDEX statement, but the first eight characters of that name must be the same as in the data set’s *psname*.)

   **y0001** Instance qualifier for the data set.

   Define one data set for the table space or index with a value of I for *y* if one of the following conditions is true:
   - You plan to run REORG with SHRLEVEL CHANGE or SHRLEVEL REFERENCE without the FASTSWITCH YES option.

- You *do not* plan to run REORG with SHRLEVEL CHANGE or SHRLEVEL REFERENCE.

Define two data sets if you plan to run REORG, using the FASTSWITCH YES option, with SHRLEVEL CHANGE or SHRLEVEL REFERENCE. Define one data set with a value of I for *y*, and one with a value of J for *y*.

For more information on defining data sets for REORG, see Part 2 of *DB2 Utility Guide and Reference*.

**nnn** Data set number. For partitioned table spaces, the number is 001 for the first partition, 002 for the second, and so forth, up to the maximum of 254 partitions.

For a nonpartitioning index on a partitioned table space that you define using the LARGE option, the maximum data set number is 128.

For simple or segmented table spaces, the number is 001 for the first data set. When little space is available, DB2 issues a warning message. If the size of the data set for a simple or a segmented table space approaches the maximum limit, define another data set. Give the new data set the same name as the first data set and the number 002. The next data set will be 003, and so on.

You can reach the extent limit for a data set before you reach the limit for a partitioned or a nonpartitioned table space. If this happens, DB2 does not extend the data set.

For detailed information about limits in DB2 for OS/390 and z/OS, see Appendix A of *DB2 Utility Guide and Reference*.

3. Use the DEFINE CLUSTER command to define the size of the primary and secondary extents of the VSAM cluster. If you specify zero for the secondary extent size, data set extension does not occur.
4. Define the data sets as LINEAR. Do not use RECORDSIZE or CONTROLINTERVALSIZE; these attributes are invalid.
5. Use the REUSE option. You must define the data set as REUSE before running the DSN1COPY utility.
6. Use SHAREOPTIONS(3,3).

The DEFINE CLUSTER command has many optional parameters that do not apply when DB2 uses the data set. If you use the parameters SPANNED, EXCEPTIONEXIT, SPEED, BUFFERSPACE, or WRITECHECK, VSAM applies them to your data set, but DB2 ignores them when it accesses the data set.

The value of the OWNER parameter for clusters that are defined for storage groups is the first SYSADM authorization ID specified at installation.

When you drop indexes or table spaces for which you defined the data sets, you must delete the data sets unless you want to reuse them. To reuse a data set, first commit, and then create a new table space or index with the same name. When DB2 uses the new object, it overwrites the old information with new information, which destroys the old data.

Likewise, if you delete data sets, you must drop the corresponding table spaces and indexes; DB2 does not do that automatically.

## DEFINE CLUSTER command

Figure 3 shows the DEFINE CLUSTER command, which defines two data sets for the SYSUSER data space. By defining both data sets for the same table space, you can run REORG with SHRLEVEL CHANGE or SHRLEVEL REFERENCE against the table space. For more information on defining data sets for REORG, see Chapter 2 of *DB2 Utility Guide and Reference*.

```
DEFINE CLUSTER-
       (NAME(DSNCAT.DSNDBC.DSNDB06.SYSUSER.I0001.A001) -
        LINEAR                                         -
        REUSE                                          -
        VOLUMES(DSNV01)                                -
        RECORDS(100 100)                               -
        SHAREOPTIONS(3 3) )                            -
      DATA                                             -
       (NAME(DSNCAT.DSNDBD.DSNDB06.SYSUSER.I0001.A001) -
    CATALOG(DSNCAT)
DEFINE CLUSTER-
       (NAME(DSNCAT.DSNDBC.DSNDB06.SYSUSER.J0001.A001) -
        LINEAR                                         -
        REUSE                                          -
        VOLUMES(DSNV01)                                -
        RECORDS(240 120)                               -
        SHAREOPTIONS(3 3) )                            -
      DATA                                             -
       (NAME(DSNCAT.DSNDBD.DSNDB06.SYSUSER.J0001.A001) -
    CATALOG(DSNCAT)
```

*Figure 3. Defining data sets for the SYSUSER table space*

For more information about defining and managing VSAM data sets, see *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

## Deferring allocation of data sets for table spaces

When you execute a CREATE TABLESPACE statement with the USING STOGROUP clause, DB2 generally defines the necessary VSAM data sets for the table space. In some cases, however, you might want to define a table space without immediately allocating the associated data sets.

For example, you might be installing a software program that requires that many table spaces be created, but your company might not need to use some of those table spaces; you might prefer not to allocate data sets for the table spaces you will not be using.

To defer the physical allocation of DB2-managed data sets, you use the DEFINE NO clause. When you specify the DEFINE NO clause, the table space is defined, but DB2 does not allocate the associated data sets until a row is inserted or loaded into a table in that table space. The DB2 catalog table SYSIBM.SYSTABLESPART contains a record of the created table space and an indication that the data sets are not yet allocated.

If you specify DEFINE NO for a table space that has a table that contains data, DB2 ignores the DEFINE NO clause and immediately allocates the storage for the table space. The DEFINE NO clause is not allowed for LOB table spaces, for table spaces in a work file database or a TEMP database, or for user-managed data sets (which are defined with the USING VCAT clause).

Using the DEFINE NO clause is recommended when:
- Performance of the CREATE TABLESPACE statement is important.
- Disk resource is constrained.

Do not use the DEFINE NO clause on a table space if you use a program outside of DB2 to propagate data into a table in the table space. The DB2 catalog stores information about whether the data sets for a table space have been allocated. If you use DEFINE NO on a table space that includes a table into which data is propagated from a program outside of DB2, the table space data sets will be allocated, but the DB2 catalog won’t reflect this fact. As a result, DB2 will act as if the data sets for the table space have not yet been allocated. The resulting inconsistency causes DB2 to deny application programs access to the data until the inconsistency is resolved.

## Managing your DB2 data sets with DFSMShsm™

The Hierarchical Storage Management functional component (DFSMShsm) of DFSMS manages space and data availability among the storage devices in your system. You can use DFSMShsm to move data sets that have not been recently used to slower, less expensive storage devices; this helps to ensure that disk space is managed efficiently.

### Recalling archive logs

DFSMShsm can automatically migrate and recall archive log and image copy data sets. If DB2 needs an archive log data set or an image copy data set that DFSMShsm has migrated, a recall begins automatically and DB2 waits for the recall to complete before continuing.

For processes that read more than one archive log data set, such as the RECOVER utility, DB2 anticipates a DFSMShsm recall of migrated archive log data sets. When a DB2 process finishes reading one data set, it can continue with the next data set without delay, because the data set might already have been recalled by DFSMShsm.

If you accepted the default value YES for the RECALL DATABASE parameter on the Operator Functions panel (DSNTIPO), DB2 also recalls migrated table spaces and index spaces. At data set open time, DB2 waits for DFSMShsm to perform the recall. The amount of time DB2 waits while the recall is being performed is specified on the RECALL DELAY parameter, which is also on panel DSNTIPO. If RECALL DELAY is set to zero, DB2 does not wait, and the recall is performed asynchronously.

DB2 subsystem data sets, including the DB2 catalog, DB2 directory, active logs, and work file databases (DSNDB07 in a non data-sharing environment), can be archived by System Managed Storage (SMS) but should be recalled by using DFSMShsm before starting DB2. An alternative is to avoid migrating by assigning a management class to these data sets that prevents migration. Considerations for using DFSMShsm for archive log data sets are discussed in “Archive log data sets” on page 336.

If a volume has a STOGROUP specified, it must be recalled only to volumes of the same device type as others in the STOGROUP.

In addition, you must coordinate the DFSMShsm automatic purge period, the DB2 log retention period, and MODIFY utility usage. Otherwise, the image copies or logs you might need during a recovery could already have been deleted.

## Migrating to DFSMShsm

If you decide to use DFSMShsm for your DB2 data sets, you should develop a migration plan with your system administrator. With user-managed data sets, you can specify DFSMShsm classes on the access method services DEFINE statement. With DB2 storage groups, you need to develop automatic class selection routines.

**General-use Programming Interface**

To allow DFSMShsm to manage your DB2 storage groups, you can use one or more asterisks as volume IDs in your CREATE STOGROUP or ALTER STOGROUP statement, as shown here:

```
CREATE STOGROUP G202
VOLUMES ('*')
VCAT DB2SMST;
```

**End of General-use Programming Interface**

This example causes all database data set allocations and definitions to use nonspecific selection through DFSMShsm filtering services.

When you use DFSMShsm and DB2 storage groups, you can use the system parameters SMSDCFL and SMSDCIX to assign table spaces and indexes to different DFSMShsm data classes.

- SMSDCFL specifies a DFSMShsm data class for table spaces. If you assign a value to SMSDCFL, DB2 specifies that value when it uses Access Method Services to define a data set for a table space.
- SMSDCIX specifies a DFSMShsm data class for indexes. If you assign a value to SMSDCIX, DB2 specifies that value when it uses Access Method Services to define a data set for an index.

Before you set the data class system parameters, you need to do two things:

- Define the data classes for your table space data sets and index data sets.
- Code the SMS automatic class selection (ACS) routines to assign indexes to one SMS storage class and to assign table spaces to a different SMS storage class.

For more information about creating data classes, see *DFSMS/MVS Storage Management Library: Implementing System-Managed Storage*.

## Using DFSMShsm with the RECOVER utility

The RECOVER utility can execute the DFDSS command RESTORE, which generally uses extensions larger than the data set’s primary and secondary values. RECOVER executes this command if the recoverable point is a copy that was taken with the CONCURRENT option. However, DFDSS RESTORE extends a data set differently from DB2, so you must alter the page set to contain extents defined by DB2. Use ALTER TABLESPACE to enlarge the primary and secondary values for DB2–managed data sets, because DB2 can run out of extents when you use REORG or LOAD REPLACE (unloading and reloading the same data).

After using ALTER TABLESPACE, the new values take effect only when you use REORG or LOAD REPLACE. Using RECOVER again does not resolve the extent definition.

For user-defined data sets, define the data sets with larger primary and secondary values (see “Managing your own DB2 data sets” on page 33).

For more information about using DFSMShsm to manage DB2 data sets, see *MVS Storage Management Library: Storage Management Subsystem Migration Planning Guide* and *DFSMS/MVS: DFSMShsm Managing Your Own Data*.

## Creating EA-enabled table spaces and index spaces

DFSMS has an extended-addressability function, which is necessary to create data sets that are larger than 4 GB. Therefore, the term for page sets that are enabled for extended addressability is *EA-enabled*. You must use EA-enabled table spaces or index spaces if you specify a DSSIZE that is larger than 4 GB in the CREATE TABLESPACE statement.

To create EA-enabled page sets, you must:

1. Use SMS to manage the data sets associated with the EA-enabled page sets.
2. Associate the data sets with a *data class* (an SMS construct) that specifies the extended format and extended addressability options.

   To make this association between data sets and the data class, use an automatic class selection (ACS) routine to assign the DB2 data sets to the relevant SMS data class. The ACS routine does the assignment based on the data set name. No performance penalty occurs for having non-EA-enabled DB2 page sets assigned to this data class, too, if you would rather not have two separate data classes for DB2.

   For user-managed data sets, you can use ACS routines or specify the appropriate data class on the DEFINE CLUSTER command when you create the data set.
3. Create the partitioned or LOB table space with a DSSIZE of 8 GB or greater. The partitioning index for the partitioned table space takes on the EA-enabled attribute from its associated table space. See *DB2 SQL Reference* for more information about the correct syntax.

   After a page set is created, you cannot use the ALTER TABLESPACE statement to change the DSSIZE. You must drop and re-create the table space.

   Also, you cannot change the data sets of the page set to turn off the extended addressability or extended format attributes. If someone modifies the data class to turn off the extended addressability or extended format attributes, DB2 issues an error message the next time it opens the page set.

## Extending DB2-managed data sets

When the data set is created, DB2 always allocates a primary allocation space on a volume that has space available and is specified in the DB2 storage group. Any new extension always gets a secondary allocation space. When the extensions reach the end of the volume, DB2 accesses all candidate volumes from the DB2 storage group and issues the access method services command ALTER ADDVOLUMES to add all volumes in the integrated catalog as candidate volumes for the data set. DB2 then makes a request to extend a secondary allocation space on any one of the candidate volumes that has space available. After the extension is successful, DB2 issues the access method services command ALTER REMOVEVOLUMES to remove all candidate volumes from the integrated catalog for the data set.

DB2 extends data sets when:
- The requested space exceeds the remaining space
- 10 percent of the smaller allocation space (but not over 10 allocation units such as tracks or cylinders) exceeds the remaining space

If DB2 fails to extend a data set with a secondary allocation space because there is no secondary allocation space available on any single candidate volume of a DB2 storage group, DB2 tries again to extend with the requested space, if the requested space is smaller than the secondary allocation space. Use IFCID 258 in statistics class 3 to monitor data set extension activity.

***Extending nonpartitioned spaces:*** For a nonpartitioned table space or an index space, DB2 defines the first piece of the page set starting with a primary allocation space, and extends that piece with secondary allocation spaces. When the end of the first piece is reached, DB2 defines a new piece (which is a new data set) and extends to that new piece starting with a primary allocation space.

***Extending partitioned spaces:*** For a partitioned table space or an index space, each partition is a data set; therefore, DB2 defines each partition with the primary allocation space and extends each partition’s data set with secondary allocation space, as needed.

***When data extension fails:*** If a data set uses all possible extents, DB2 cannot extend that data set. For a partitioned page set, the extension fails only for the particular partition that DB2 is trying to extend. For nonpartitioned page sets, DB2 cannot extend to a new data set piece, which means the extension for the entire page set fails.

To avoid extension failures, the value of (PRIQTY + *max_extents* × SECQTY) must be at least as large as the data set size (as specified on the DSSIZE clause or the implicit size for that type of page set). For nonpartitioning indexes, that value must reach the value for PIECESIZE (explicitly or implicitly specified). If DB2 reaches the maximum number of extents before reaching the limit, the extension fails.

## Extending user-managed data sets

User-managed data sets are extended using only volumes available in the integrated catalog facility catalog. Before the current volume runs out of space, you must issue the access method services commands ALTER ADDVOLUMES or ALTER REMOVEVOLUMES for candidate volumes.

# Chapter 5. Implementing your design

The information in this chapter is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

This chapter provides information on advanced topics related to implementing a database:

Table 7 shows where you can find more information about topics related to implementing a database design.

*Table 7. More information about implementing a database design*

| For more information about... | See... |
|---|---|
| Basic concepts in implementing a database design for DB2 Universal Database for OS/390 and z/OS, including:<br>• Choosing names for DB2 objects<br>• Implementing your databases<br>• Implementing your table spaces, including reorganizing your data<br>• Implementing your tables<br>• Implementing your indexes<br>• Implementing referential constraints<br>• Implementing your views | *An Introduction to DB2 for OS/390* |
| Details on SQL statements used to implement a database design (CREATE and DECLARE, for example) | *DB2 SQL Reference* |
| Loading tables with referential constraints | *DB2 Utility Guide and Reference* |
| Using the catalog in database design | Appendix E of *DB2 SQL Reference* |

## Implementing your databases

In DB2 for OS/390 and z/OS, a database is a logical collection of tables spaces and index spaces. Consider the following factors when deciding whether to define a new database for a new set of objects:

- An entire database can be started and stopped as a unit; the statuses of all its objects can be displayed by a single command that names only the database. Therefore, place a set of tables that are used together into the same database. (The same database holds all indexes on those tables.)
- Some operations lock an entire database. For example, some phases of the LOAD utility prevent some SQL statements from using the same database concurrently. Therefore, placing many unrelated tables in a single database is often inconvenient.

  When one user is executing a CREATE, ALTER, or DROP statement for a table, no other user can access the database that contains that table. QMF™ users, especially, might do a great deal of data definition; the operations SAVE DATA and ERASE *data-object* are accomplished by creating and dropping DB2 tables. For maximum concurrency, create a separate database for each QMF user.

- The internal database descriptors (DBDs) might become inconveniently large; Part 2 of *DB2 Installation Guide* contains some calculations showing how the size depends on the number of columns in a table. DBDs grow as new objects are defined, but they do not immediately shrink when objects are dropped—the DBD space for a dropped object is not reclaimed until the MODIFY RECOVERY utility is used to delete records of obsolete copies from SYSIBM.SYSCOPY. DBDs occupy storage and are the objects of occasional input and output operations. Therefore, limiting the size of DBDs is another reason to define new databases. The MODIFY utility is described in Part 2 of *DB2 Utility Guide and Reference*.

If you are using declared temporary tables, you must define a database that is defined AS TEMP (the TEMP database). DB2 stores all declared temporary tables in the TEMP database. The majority of the factors described above do not apply to the TEMP database. For details on declared temporary tables, see “Distinctions between DB2 base tables and temporary tables” on page 45.

## Implementing your table spaces

Table spaces are the physical spaces that hold tables. A table space can have one or more tables. Simple and segmented table spaces hold a maximum of 64 GB of data and might use one or more VSAM data sets. Partitioned table spaces that are created with the DSSIZE or LARGE option, and LOB table spaces can be larger. Table spaces are divided into units called pages that are either 4 KB, 8 KB, 16 KB, or 32 KB in size. As a general rule, have no more than 50 to 100 table spaces in one DB2 database.

You need to create additional table spaces if your database contains LOB data. For more information, see Chapter 5 of *DB2 SQL Reference*.

Data in most table spaces can be compressed, which can allow you to store more data on each data page. For more information, see “Compressing your data” on page 606.

### Creating a table space explicitly

Use the CREATE TABLESPACE statement to create a table space explicitly. The statement allows you to specify the attributes of the table space.

Generally when you use the CREATE TABLESPACE statement with the USING STOGROUP clause, DB2 allocates data sets for the table space. However, if you also specify the DEFINE NO clause, you can defer the allocation of data sets until data is inserted or loaded into a table in the table space. For more information about deferring data set allocation, see “Deferring allocation of data sets for table spaces” on page 36.

You can create simple, segmented, partitioned, and LOB table spaces. For detailed information about CREATE TABLESPACE, see Chapter 5 of *DB2 SQL Reference*.

### Creating a table space implicitly

As with DB2 storage groups and databases, you do not need to create a table space before you create a table unless you are defining a declared temporary table or managing all your own data sets. When you use CREATE TABLE, DB2 generates a table space for you. However, DB2 generates a table space only if you use CREATE TABLE without specifying an existing table space name. If the table contains a LOB column and SQLRULES are STD, DB2 also creates the LOB table space, the auxiliary table, and auxiliary index. If you do not specify a database name in the CREATE TABLE statement, DB2 uses the default database, DSNDB04, and the default DB2 storage group, SYSDEFLT. DB2 also uses defaults for space allocation and other table space attributes.

If you create a table space implicitly, DB2 derives a table space name from the name of your table according to these rules:

- The table space name is the same as the table name if these conditions apply:
  – No other table space or index space in the database already has that name.
  – The table name has no more than eight characters.
  – The characters are all alphanumeric, and the first character is not a digit.
- If some other table space in the database already has the same name as the table, DB2 assigns a name of the form *xxxxnyyy*, where *xxxx* is the first four characters of the table name, and *nyyy* is a single digit and three letters that guarantees uniqueness.

DB2 stores this name in the DB2 catalog in the SYSIBM.SYSTABLESPACE table along with all your other table space names. The rules for LOB table spaces are in Chapter 5 of *DB2 SQL Reference*.

## Choosing a page size

DB2 provides many options for data page sizes. The size of the data page is determined by the buffer pool in which you define the table space. For example, a table space that is defined in a 4-KB buffer pool has 4-KB page sizes, and one that is defined in an 8-KB buffer pool has 8-KB page sizes. (Indexes must be defined in a 4-KB buffer pool.)

Data in table spaces is stored and allocated in 4-KB record segments. Thus, an 8-KB page size means two 4-KB records, and a 32-KB page size means eight 4-KB records. A good starting point is to use the default of 4-KB page sizes when access to the data is random and only a few rows per page are needed. If row sizes are very small, using the 4-KB page size is recommended.

However, there are situations in which larger page sizes are needed or recommended:

- **When the size of individual rows is greater than 4 KB.** In this case, you must use a larger page size. When considering the size of work file table spaces, remember that some SQL operations, such as joins, can create a result row that does not fit in a 4-KB page. That is a good reason to have at least one work file that has 32-KB pages. (Work files cannot use 8-KB or 16-KB pages.)
- **When you can achieve higher density on disk by choosing a larger page size.** For example, only one 2100-byte record can be stored in a 4-KB page, which wastes almost half of the space. However, storing the record in a 32-KB page can significantly reduce this waste. The downside with this approach is the potential of incurring higher buffer pool storage costs or higher I/O costs—if you only touch a small number of rows, you are bringing a bigger chunk of data from disk into the buffer pool.

  Using 8-KB or 16-KB page sizes can let you store more data on your disk with less impact on I/O and buffer pool storage costs. If you use a larger page size and access is random, you might need to go back and increase the size of the buffer pool to achieve the same read-hit ratio you do with the smaller page size.
- **When a larger page size can reduce data sharing overhead.** One way to reduce the cost of data sharing is to reduce the number of times the coupling facility must be accessed. Particularly for sequential processing, larger page

sizes can reduce this number. More data can be returned on each access of the coupling facility, and fewer locks must be taken on the larger page size, further reducing coupling facility interactions.

If data is returned from the coupling facility, each access that returns more data is more costly than those that return smaller amounts of data, but, because the total number of accesses is reduced, coupling facility overhead is reduced.

For random processing, using an 8-KB or 16-KB page size instead of a 32-KB page size might improve the read-hit ratio to the buffer pool and reduce I/O resource consumption.

## Choosing a page size for LOBs

Choosing a page size for LOBs (in the LOB table space) is a tradeoff between minimizing the number of getpages (maximizing performance) and not wasting space. With LOB table spaces, no more than one LOB value is ever stored in a given page in a LOB table space. Space that is not used by the LOB value in the last page that is occupied by the LOB remains unused. DB2 also uses additional space for control information. The smaller the LOB, the greater the proportion of space for this “non-data” is used.

For example, if you have a 17-KB LOB, the 4-KB page size is the most efficient for storage. A 17-KB LOB requires five 4-KB pages for a total of 20 KB of storage space. Pages that are 8 KB, 16 KB, and 32 KB in size waste more space, because they require 24 KB, 32 KB, and 32 KB, respectively, for the LOB.

Table 8 shows that the number of data pages is lower for larger page sizes, but larger page sizes might have more unused space.

*Table 8. Relationship between LOB size and data pages based on page size*

| LOB size | Page size | LOB data pages | % Non-LOB data or unused space |
|---|---|---|---|
| 262 144 bytes | 4 KB | 64 | 1.6 |
| | 8 KB | 32 | 3.0 |
| | 16 KB | 16 | 5.6 |
| | 32 KB | 8 | 11.1 |
| 4 MB | 4 KB | 1029 | 0.78 |
| | 8 KB | 513 | 0.39 |
| | 16 KB | 256 | 0.39 |
| | 32 KB | 128 | 0.78 |
| 33 MB | 4 KB | 8234 | 0.76 |
| | 8 KB | 4106 | 0.39 |
| | 16 KB | 2050 | 0.19 |
| | 32 KB | 1024 | 0.10 |

***Choosing a page size based on average LOB size:*** If you know all of your LOBs are not the same size, you can still make an estimate of what page size to choose. To estimate the average size of a LOB, you need to add a percentage to account for unused space and control information. To estimate the average size of a LOB value, use the following formula:

```
LOB size = (average LOB length) × 1.05
```

Table 9 has some suggested page sizes for LOBs with the intent to reduce the amount of I/O (getpages).

*Table 9. Suggested page sizes based on average LOB length*

| Average LOB size (n) | Suggested page size |
|---|---|
| n ≤ 4 KB | 4 KB |
| 4 KB < n ≤ 8 KB | 8 KB |
| 8 KB < n ≤ 16 KB | 16 KB |
| 16 KB < n | 32 KB |

The estimates in Table 9 mean that a LOB value of 17 KB can mean 15 KB of unused space. Again, you must analyze your data to determine what is best.

***General guidelines for LOBs of same size:*** If your LOBs are all the same size, you can fairly easily choose a page size that uses space efficiently without sacrificing performance. For LOBs that are all the same size, consider the alternative in Table 10 to maximize your space savings.

*Table 10. Suggested page sizes when LOBs are same size*

| LOB size (y) | Suggested page size |
|---|---|
| y ≤ 4 KB | 4 KB |
| 4 KB < y ≤ 8 KB | 8 KB |
| 8 KB < y ≤ 12 KB | 4 KB |
| 12 KB < y ≤ 16 KB | 16 KB |
| 16 KB < y ≤ 24 KB | 8 KB |
| 24 KB < y ≤ 32 KB | 32 KB |
| 32 KB < y ≤ 48 KB | 16 KB |
| 48 KB < y | 32 KB |

## Distinctions between DB2 base tables and temporary tables

Table 11 summarizes important differences between base tables and the two types of temporary tables. Additional examples of implementing temporary tables and information about restrictions and extensions of temporary tables can be found in Part 1 of *DB2 Application Programming and SQL Guide* and in Chapter 5 of *DB2 SQL Reference*. For information about temporary tables and their impact on DB2 resources, see “Work file data sets” on page 599.

*Table 11. Important distinctions between DB2 base tables and DB2 temporary tables*

| Base tables | Created temporary tables | Declared temporary tables |
|---|---|---|
| **Creation, persistence, and ability to share table descriptions** | | |

*Table 11. Important distinctions between DB2 base tables and DB2 temporary tables (continued)*

| Base tables | Created temporary tables | Declared temporary tables |
|---|---|---|
| CREATE TABLE statement puts a description of the table in catalog table SYSTABLES. The table description is persistent and is shareable across application processes.<br>The name of the table in the CREATE statement can be a two-part or three-part name. If the table name is not qualified, DB2 implicitly qualifies the name using the standard DB2 qualification rules applied to the SQL statements. | CREATE GLOBAL TEMPORARY TABLE statement puts a description of the table in catalog table SYSTABLES. The table description is persistent and is shareable across application processes.<br>The name of the table in the CREATE statement can be a two-part or three-part name. If the table name is not qualified, DB2 implicitly qualifies the name using the standard DB2 qualification rules applied to the SQL statements. | DECLARE GLOBAL TEMPORARY TABLE statement does *not* put a description of the table in catalog table SYSTABLES. The table description is *not* persistent beyond the life of the application process that issued the DECLARE statement and the description is known only to that application process. Thus, each application process could have its own possibly unique description of the same table.<br>The name of the table in the DECLARE statement can be a two-part or three-part name. If the table name is qualified, SESSION must be used as the qualifier for the owner (the second part in a three-part name). If the table name in not qualified, DB2 implicitly uses SESSION as the qualifier. |
| **Table instantiation and ability to share data** | | |
| CREATE TABLE statement creates one empty instance of the table, and all application processes use that one instance of the table. The table and data are persistent. | CREATE GLOBAL TEMPORARY TABLE statement does *not* create an instance of the table. The first implicit or explicit reference to the table in an OPEN, SELECT, INSERT, or DELETE operation executed by any program in the application process creates an empty instance of the given table. Each application process has its own unique instance of the table, and the instance is not persistent beyond the life of the application process. | DECLARE GLOBAL TEMPORARY TABLE statement creates an empty instance of the table for the application process. Each application process has its own unique instance of the table, and the instance is not persistent beyond the life of the application process. |
| **References to the table in application processes** | | |
| References to the table name in multiple application processes refer to the *same* single persistent table description and *same* instance at the current server.<br>If the table name being referenced is not qualified, DB2 implicitly qualifies the name using the standard DB2 qualification rules applied to the SQL statements. The name can be a two-part or three-part name. | References to the table name in multiple application processes refer to the *same* single persistent table description but to a *distinct* instance of the table for each application process at the current server.<br>If the table name being referenced is not qualified, DB2 implicitly qualifies the name using the standard DB2 qualification rules applied to the SQL statements. The name can be a two-part or three-part name. | References to that table name in multiple application processes refer to a *distinct* description and instance of the table for each application process at the current server.<br>References to the table name in an SQL statement (other than the DECLARE GLOBAL TEMPORARY TABLE statement) must include **SESSION** as the qualifier (the first part in a two-part table name or the second part in a three-part name). If the table name is not qualified with **SESSION**, DB2 assumes the reference is to a base table. |
| **Table privileges and authorization** | | |

*Table 11. Important distinctions between DB2 base tables and DB2 temporary tables (continued)*

| Base tables | Created temporary tables | Declared temporary tables |
|---|---|---|
| The owner implicitly has all table privileges on the table and the authority to drop the table. The owner's table privileges can be granted and revoked, either individually or with the ALL clause.<br>Another authorization ID can access the table only if it has been granted appropriate privileges for the table. | The owner implicitly has all table privileges on the table and the authority to drop the table. The owner's table privileges can be granted and revoked, but only with the ALL clause; individual table privileges cannot be granted or revoked.<br>Another authorization ID can access the table only if it has been granted ALL privileges for the table. | PUBLIC implicitly has all table privileges on the table without GRANT authority and has the authority to drop the table. These table privileges cannot be granted or revoked.<br>Any authorization ID can access the table without a grant of any privileges for the table. |
| **Indexes and other SQL statement support** | | |
| Indexes and SQL statements that modify data (INSERT, UPDATE, DELETE, and so on) are supported. | Indexes, UPDATE (searched or positioned), and DELETE (positioned only) are *not* supported. | Indexes and SQL statements that modify data (INSERT, UPDATE, DELETE, and so on) are supported |
| **Locking, logging, and recovery** | | |
| Locking, logging, and recovery *do* apply. | Locking, logging, and recovery *do not* apply. Work files are used as the space for the table. | Some locking, logging, and limited recovery *do* apply. No row or table locks are acquired. Share-level locks on the table space and DBD are acquired. A segmented table lock is acquired when all the rows are deleted from the table or the table is dropped. Undo recovery (rolling back changes to a savepoint or the most recent commit point) is supported, but redo recovery (forward log recovery) is not supported. |
| **Table space and database operations** | | |
| Table space and database operations, *do* apply. | Table space and database operations, *do not* apply. | Table space and database operations, *do* apply. |
| **Table space requirements and table size limitations** | | |
| The table can be stored in simple table spaces in default database DSNDB04 or user-defined table spaces (simple, segmented, or partitioned) in user-defined databases.<br>Table cannot span table spaces. Therefore, the size of the table is limited by the table space size (as determined by the primary and secondary space allocation values specified for the table space's data sets) and the shared usage of the table space among multiple users. When the table space is full, an error occurs for the SQL operation. | The table is stored in table spaces in the work file database.<br>The table can span work file table spaces. Therefore, the size of the table is limited by the number of available work file table spaces, the size of each table space, and the number of data set extents that are allowed for the table spaces. Unlike the other types of tables, created temporary tables do not reach size limitations as easily. | The table is stored in segmented table spaces in the TEMP database (a database that is defined AS TEMP).<br>The table cannot span table spaces. Therefore, the size of the table is limited by the table space size (as determined by the primary and secondary space allocation values specified for the table space's data sets) and the shared usage of the table space among multiple users. When the table space is full, an error occurs for the SQL operation. |

## Using schemas

A *schema* is a collection of named objects. The objects that a schema can contain include distinct types, functions, stored procedures, and triggers. An object is assigned to a schema when it is created.

When a distinct type, function, stored procedure, or trigger is created, it is given a qualified two-part name. The first part is the schema name (or the qualifier), which is either implicitly or explicitly specified. The default schema is the authorization ID of the owner of the plan or package. The second part is the name of the object.

Schemas extend the concept of qualifiers for tables, views, indexes, and aliases to enable the qualifiers for distinct types, functions, stored procedures, and triggers to be called schema names.

You can create a schema with the schema processor by using the CREATE SCHEMA statement. CREATE SCHEMA cannot be embedded in a host program or executed interactively. To process the CREATE SCHEMA statement, you must use the schema processor, as described in “Processing schema definitions” on page 49. The ability to process schema definitions is provided for conformance to ISO/ANSI standards. The result of processing a schema definition is identical to the result of executing the SQL statements without a schema definition.

Outside of the schema processor, the order of statements is important. They must be arranged so that all referenced objects have been previously created. This restriction is relaxed when the statements are processed by the schema processor if the object table is created within the same CREATE SCHEMA. The requirement that all referenced objects have been previously created is not checked until all of the statements have been processed. For example, within the context of the schema processor, you can define a constraint that references a table that does not exist yet or GRANT an authorization on a table that does not exist yet. Figure 4 is an example of a valid schema definition.

```
CREATE SCHEMA AUTHORIZATION SMITH

CREATE TABLE TESTSTUFF
 (TESTNO   CHAR(4),
  RESULT   CHAR(4),
  TESTTYPE  CHAR(3))

CREATE TABLE STAFF
 (EMPNUM   CHAR(3) NOT NULL,
  EMPNAME  CHAR(20),
  GRADE    DECIMAL(4),
  CITY     CHAR(15))

CREATE VIEW STAFFV1
         AS SELECT * FROM STAFF
            WHERE  GRADE >= 12

GRANT INSERT ON TESTSTUFF TO PUBLIC

GRANT ALL PRIVILEGES ON STAFF
        TO PUBLIC
```

*Figure 4. Example of schema processor input*

## Authorization to process schema definitions

The schema processor sets the current SQLID to the value of the schema authorization ID before executing any of the statements in the schema definition. Therefore, that ID must have SYSADM or SYSCTRL authority, or it must be the primary or one of the secondary authorization IDs of the process that executes the schema processor. The same ID must have all the privileges that are needed to execute all the statements in the schema definition.

## Processing schema definitions

Run the schema processor (DSNHSP) as a batch job; use the sample JCL provided in member DSNTEJ1S of the SDSNSAMP library. The schema processor accepts only one schema definition in a single job. No statements that are outside the schema definition are accepted. Only SQL comments can precede the CREATE SCHEMA statement; the end of input ends the schema definition. SQL comments can also be used within and between SQL statements.

The processor takes the SQL from CREATE SCHEMA (the SYSIN data set), dynamically executes it, and prints the results in the SYSPRINT data set.

If a statement in the schema definition has an error, the schema processor processes the remaining statements but rolls back all the work at the end. In this case, you need to fix the statement in error and resubmit the entire schema definition.

# Chapter 6. Loading data into DB2 tables

This chapter provides an overview of how to load data into DB2 tables:

You can use several methods to fill DB2 tables, but you will probably load most of your tables by using the LOAD utility.

## Loading methods

You can load tables in DB2 by using:

- The LOAD utility. See “Loading tables with the LOAD utility” and Part 2 of *DB2 Utility Guide and Reference* . The utility loads data into DB2 persistent tables, from either sequential data sets or SQL/DS™ unload data sets, using BSAM. The LOAD utility cannot be used to load data into DB2 temporary tables.

  When loading tables with indexes, referential constraints, or table check constraints, LOAD can perform several checks on the validity of data. If errors are found, the table space being loaded, its index spaces, and even other table spaces might be left in a restricted status.

  Plan to make necessary corrections and remove restrictions after any such LOAD job. For instructions, see “Replacing data” on page 52.
- An SQL INSERT statement in an application program. See “Loading data using the SQL INSERT statement” on page 53 and *DB2 SQL Reference*. This method allows you to develop an application that loads data into DB2 tables that is tailored to your own requirements.
- An SQL INSERT statement to copy all or selected rows of another table. You can do that interactively, using SPUFI. See “Loading data using the SQL INSERT statement” on page 53 and *DB2 SQL Reference*.

To reformat data from IMS DL/I databases and VSAM and SAM loading for the LOAD utility, use DB2 DataPropagator. See “Loading data from DL/I” on page 54.

For general guidance about running DB2 utility jobs, see *DB2 Utility Guide and Reference*. For information about DB2 DataPropagator, see *DB2 UDB Replication Guide and Reference*.

## Loading tables with the LOAD utility

Use LOAD to load one or more persistent tables of a table space, or one or more partitions of a table space. LOAD operates on a table space, so you must have authority for all tables in the table space when you run LOAD.

The LOAD utility 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.

You can load input data into ASCII, EBCDIC, or UNICODE tables. The ASCII, EBCDIC, and UNICODE options on the LOAD utility statement let you specify whether the data in the input file is ASCII, EBCDIC, or UNICODE. The CCSID option of the LOAD utility statement lets you specify the CCSIDs of the data in the input file. If the CCSID of the input data does not match the CCSID of the table space, the input fields are converted to the CCSID of the table space before they are loaded.

For nonpartitioned table spaces, or if nonpartitioning indexes are defined on a table in a partitioned table space, data in the table space being loaded is unavailable to other application programs during the load operation. Also, some SQL statements, such as CREATE, DROP, and ALTER, might experience contention when they run against another table space in the same DB2 database while the table is being loaded.

Additionally, LOAD can be used to:
- Compress data and build a compression dictionary
- Convert data between compatible data types
- Load multiple tables in a single table space

When you load a table and do not supply a value for one or more of the columns, the action DB2 takes depends on the circumstances.
- If the column is not a ROWID or identity column, DB2 loads the default value of the column, which is specified by the DEFAULT clause of the CREATE or ALTER TABLE statement.
- If the column is a ROWID or identity column that uses the GENERATED BY DEFAULT option, DB2 provides a unique value.

For ROWID or identity columns that use the GENERATED ALWAYS option, you cannot supply a value, because this option means that DB2 always generates a unique value.

The LOAD utility treats LOB columns as varying-length data. The length value for a LOB column must be 4 bytes. When the input record is greater than 32 KB, you might have to load the LOB data separately. The auxiliary tables are loaded when the base table is loaded. You cannot specify the name of the auxiliary table to load.

## Replacing data

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).

***Making corrections after LOAD:*** LOAD can place a table space or index space into one of several kinds of restricted status. Your use of a table space in restricted status is severely limited. In general, you cannot access its data through SQL; you can only drop the table space or one of its tables, or perform some operation that resets the status.

To discover what spaces are in restricted status, use the command:

```
-DISPLAY DATABASE (*) SPACENAM (*) RESTRICT
```

LOAD places a table space in the copy-pending state if you load with LOG NO, which you might do to save space in the log. Immediately after that operation, DB2 cannot recover the table space. However, the table space can be recovered by loading it again. Prepare for recovery, and remove 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.)

When you use REORG or LOAD REPLACE with the COPYDDN keyword, a full image copy data set (SHRLEVEL REF) is created during the execution of the REORG or LOAD utility. This full image copy is known as an *inline copy*. The table space is not left in copy-pending state regardless of which LOG option was specified for the utility.

The inline copy is valid only if you replace the entire table space or partition. If you request an inline copy by specifying the keyword COPYDDN in a LOAD utility statement, but the load is RESUME YES, or is RESUME NO and REPLACE is not specified, an error message is issued and the LOAD terminates.

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

LOAD places a table space in the check-pending status if its referential or check integrity is in doubt. Because of this restriction, use of the CHECK DATA utility is recommended. That utility locates and, optionally, removes invalid data. If the CHECK DATA utility removes invalid data, the data remaining satisfies all referential and table check constraints, and the check-pending restriction is lifted.

## Loading data using the SQL INSERT statement

The information in this section, up to “Loading data from DL/I” on page 54 is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

Another way to load data into tables is with the SQL INSERT statement. You can issue the statement interactively or embed it in an application program.

The simplest form of INSERT inserts a single row of data. In this form of the statement, you specify the table name, the columns into which the data is to be inserted, and the data itself.

Suppose you create a test table, TEMPDEPT, with the same characteristics as the department table:

```
CREATE TABLE SMITH.TEMPDEPT
  (DEPTNO  CHAR(3)  NOT NULL,
  DEPTNAME VARCHAR(36)   NOT NULL,
  MGRNO    CHAR(6)  NOT NULL,
  ADMRDEPT CHAR(3)  NOT NULL)
  IN DSN8D71A.DSN8S71D;
```

To add a row to table TEMPDEPT, you can enter:

```
INSERT INTO SMITH.TEMPDEPT
  VALUES ('X05','EDUCATION','000631','A01');
```

If you write an application program to load data into tables, you use that form of INSERT, probably with host variables instead of the actual values shown above.

You can also use a form of INSERT that copies rows from another table. You can load TEMPDEPT with the following statement:

```
INSERT INTO SMITH.TEMPDEPT
  SELECT DEPTNO,DEPTNAME,MGRNO,ADMRDEPT
  FROM DSN8710.DEPT
  WHERE ADMRDEPT='D01';
```

The statement loads TEMPDEPT with data from the department table about all departments that report to Department D01.

If you are inserting a large number of rows, then consider using one of the following methods:

- Use the LOAD or UNLOAD utilities.
- Use multiple INSERT statements with predicates that isolate the data to be loaded, and then commit after each insert operation.

When a table, whose indexes are already defined, is populated by using the INSERT statement, both the FREEPAGE and the PCTFREE parameters are ignored. FREEPAGE and PCTFREE are only in effect during a LOAD or REORG operation.

**Tables with ROWID columns:** You can load a value for a ROWID column with an INSERT and fullselect only if the ROWID column is defined as GENERATED BY DEFAULT. If you have a table with a column defined as ROWID GENERATED ALWAYS, you can propagate non-ROWID columns from a table with the same definition.

For the complete syntax of the INSERT statement, see *DB2 SQL Reference*.

## Loading data from DL/I

To convert data in IMS DL/I databases from a hierarchical structure to a relational structure so that it can be loaded into DB2 tables, you can use the DataRefresher™ licensed programs.

# Chapter 7. Altering your database design

The information in this chapter is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

After using a relational database for a while, you might want to change some aspects of its design. This chapter tells how to change:

- The definitions of DB2 objects; see:
  - “Altering DB2 storage groups” on page 56
  - “Altering DB2 databases” on page 57
  - “Altering table spaces” on page 57
  - “Altering tables” on page 59
  - “Altering indexes” on page 69
  - “Altering views” on page 70
- Data set high-level qualifier; see “Changing the high-level qualifier for DB2 data sets” on page 71.
- The location of DB2 data; see “Moving DB2 data” on page 78.

You can alter the definition of a DB2 object by using one of the following methods:

- Using an SQL ALTER statement
- Dropping the object and then re-creating it with different specifications

## Using the ALTER statement

Use the SQL ALTER statement to change DB2 storage groups, databases, table spaces, tables, and indexes. ALTER changes the way those objects are defined in the DB2 catalog, but it does not accomplish every change. For example, you cannot drop a column from a table using the ALTER statement. Application and object registration tables can restrict the use of ALTER. See “Chapter 11. Controlling access through a closed application” on page 157 for more information.

## Dropping and re-creating DB2 objects

When you cannot make a change with ALTER, you must:

1. Use the DROP statement to remove the object.
2. Use the COMMIT statement to commit the changes to the database object.
3. Use the CREATE statement to re-create the object.

The DROP statement has a cascading effect; objects that are dependent on the dropped object are also dropped. For example, all authorities for those objects disappear. Plans or packages that reference deleted objects are marked invalid by DB2. Before dropping an object, check the DB2 catalog to determine the impact of the operation.

When a user with the EXECUTE authority tries to execute an invalidated plan or package, DB2 first rebinds it automatically, using the same options that were used during the most recent bind operation. (To see if a plan or package is invalidated, check the VALID column in SYSIBM.SYSPLAN or SYSIBM.SYSPACKAGE.)

For more information about invalidated plans and packages and rebinding, see Part 4 of *DB2 Application Programming and SQL Guide*. For more information about dropping a table, see “Implications of dropping a table” on page 66. Information about dropping other objects is in *DB2 SQL Reference*.

## Altering DB2 storage groups

You can use the ALTER STOGROUP statement to specify whether you want SMS to manage your DB2 storage groups, or to add or remove volumes from a storage group. If you want to migrate to another device type or change the catalog name of the integrated catalog facility, you need to move the data. See “Moving DB2 data” on page 78 for more information.

## Letting SMS manage your DB2 storage groups

To let SMS manage the storage needed for the objects that the storage group supports, specify ADD VOLUMES ('*') and REMOVE VOLUMES (*current-vols*) in the ALTER STOGROUP statement, where *current-vols* is the list of the volumes currently assigned to the storage group:

```
ALTER STOGROUP DSN8G710
  REMOVE VOLUMES (VOL1)
  ADD VOLUMES ('*');
```

SMS manages every new data set that is created after the ALTER STOGROUP statement is executed; SMS does not manage data sets that are created before the execution of the statement. See “Migrating to DFSMShsm” on page 38 for more considerations for using SMS to manage data sets.

## Adding or removing volumes from a DB2 storage group

When you add or remove volumes from a storage group, all the volumes in a storage group must be of the same type; and, when a storage group is used to extend a data set, the volumes must have the same device type as the volumes that were used when the data set was defined.

The changes you make to the volume list by ALTER STOGROUP have no effect on existing storage. Changes take effect when new objects are defined or when the REORG, RECOVER, or LOAD REPLACE utilities are used on those objects. For example, if you use ALTER STOGROUP to remove volume 22222 from storage group DSN8G710, the DB2 data on that volume remains intact. However, when a new table space is defined using DSN8G710, volume 22222 is not available for space allocation.

To force a volume off and add a new volume, follow these steps:

1. Use the SYSIBM.SYSTABLEPART catalog table to determine which table spaces are associated with the storage group. The following query tells which table spaces use storage group DSN8G710:

   ```
   SELECT TSNAME, DBNAME
     FROM SYSIBM.SYSTABLEPART
     WHERE STORNAME ='DSN8G710' AND STORTYPE = 'I';
   ```

2. Make an image copy of each table space; for example, COPY TABLESPACE *dbname.tsname* DEVT SYSDA.
3. Ensure that the table space is not being updated in such a way that the data set might need to be extended. For example, you can stop the database.
4. Use the ALTER STOGROUP statement to remove the volume associated with the old storage group and to add the new volume.

   ```
   ALTER STOGROUP DSN8G710
     REMOVE VOLUMES (VOL1)
     ADD VOLUMES (VOL2);
   ```

**Important:** When a new volume is added, or when a storage group is used to extend a data set, the volumes must have the same device type as the volumes used when the data set was defined.

5. Start the database with utility-only processing, and use the RECOVER or REORG utility to move the data in each table space; for example, RECOVER *dbname.tsname*.
6. Start the database.

## Altering DB2 databases

The ALTER DATABASE statement allows you to change the following clauses that are used to create a database:

- STOGROUP. Lets you change the name of the default storage group to support disk space requirements for table spaces and indexes within the database. The new default DB2 storage group is used only for new table spaces and indexes; existing definitions do not change.
- BUFFERPOOL. Lets you change the name of the default buffer pool for table spaces and indexes within the database. Again, it applies only to new table spaces and indexes; existing definitions do not change.
- INDEXBP. Lets you change the name of the default buffer pool for the indexes within the database. The new default buffer pool is used only for new indexes; existing definitions do not change.

## Altering table spaces

Use the ALTER TABLESPACE statement to modify the description of a table space. The statement can be embedded in an application program or issued interactively. For details on the ALTER TABLESPACE statement, see Chapter 5 of *DB2 SQL Reference*.

### Changing the space allocation for user-managed data sets

If the table space is supported by user-managed data sets, use this method to change the space allocation:

1. Run the REORG utility, and specify the UNLOAD PAUSE option.
2. When the utility has completed the unload and has stopped, delete and redefine the data sets.

   If the table space was created with the CLOSE NO parameter, the table space must be stopped with the STOP DATABASE command and the SPACENAM option before you delete and define the data sets.
3. Resubmit the utility job with the RESTART(PHASE) parameter specified on the EXEC statement. The job now uses the new data sets when reloading.

Use of the REORG utility to extend data sets causes the newly acquired free space to be distributed throughout the table space rather than to be clustered at the end.

### Dropping, re-creating, or converting a table space

To make changes to a table space, such as changing SEGSIZE or the number of partitions or to convert it to a large table space, you must first drop the table space and then re-create it. You must commit the DROP TABLESPACE statement before creating a table space or index with the same name. When you drop a table space, all entries for that table space are dropped from SYSIBM.SYSCOPY. This makes recovery for that table space impossible from previous image copies. You can change or convert your table spaces with the following steps:

1. Locate the original CREATE TABLE statement and all authorization statements for all tables in the table space. (For example, TA1, TA2, TA3, ... in TS1.) If you cannot find these statements, query the DB2 catalog to determine the table's description, the description of all indexes and views on it, and all users with privileges on the table.
2. In another table space (TS2, for example), create tables TB1, TB2, TB3, ... identical to TA1, TA2, TA3, .... For example, use statements like:

   ```
   CREATE TABLE TB1 LIKE TA1 IN TS2;
   ```
3. If necessary, unload the data using a statement such as:

   ```
   REORG TABLESPACE DSN8D71A.TS1 LOG NO SORTDATA UNLOAD EXTERNAL;
   ```

   Or, you can insert the data from your old tables into the new tables by executing an INSERT statement for each table. For example:

   ```
   INSERT INTO TB1
     SELECT * FROM TA1;
   ```

   If a table contains a ROWID column or an identity column and you want to keep the existing column values, you must define that column as GENERATED BY DEFAULT. If the ROWID column or identity column is defined with GENERATED ALWAYS, and you want DB2 to generate new values for that column, specify OVERRIDING USER VALUE on the INSERT statement with the subselect.
4. Drop the table space by executing the statement:

   ```
   DROP TABLESPACE TS1;
   ```

   The compression dictionary for the table space is dropped, if one exists. All tables in TS1 are dropped automatically.
5. Commit the DROP statement.
6. Create the new table space, TS1, and grant the appropriate user privileges. You can also create a partitioned table space. You could use the following statements:

   ```
   CREATE TABLESPACE TS1
      IN DSN8D71A
      USING STOGROUP DSN8G710
        PRIQTY 4000
        SECQTY 130
        ERASE NO
    NUMPARTS 95
     (PART 45 USING STOGROUP DSN8G710
        PRIQTY 4000
        SECQTY 130
        COMPRESS YES,
      PART 62 USING STOGROUP DSN8G710
        PRIQTY 4000
        SECQTY 130
        COMPRESS NO)
    LOCKSIZE PAGE
    BUFFERPOOL BP1
    CLOSE NO;
   ```
7. Create new tables TA1, TA2, TA3, ....
8. Re-create indexes on the tables, and re-grant user privileges on those tables. See “Implications of dropping a table” on page 66 for more information.
9. Execute an INSERT statement for each table. For example:

   ```
   INSERT INTO TA1
     SELECT * FROM TB1;
   ```

If a table contains a ROWID column or an identity column and you want to keep the existing column values, you must define that column as GENERATED BY DEFAULT. If the ROWID column or identity column is defined with GENERATED ALWAYS, and you want DB2 to generate new values for that column, specify OVERRIDING USER VALUE on the INSERT statement with the subselect.

10. Drop table space TS2.

    If a table in the table space has been created with RESTRICT ON DROP, you must alter that table to remove the restriction before you can drop the table space.

11. Notify users to re-create any synonyms they had on TA1, TA2, TA3, ....

## Altering tables

When you alter a table, you do not change the data in the table; you merely change the specifications you used in creating the table.

## Using the ALTER TABLE statement

With ALTER TABLE you can:

- Add a new column; see “Adding a new column”.
- Change the AUDIT clause, using the options ALL, CHANGES, or NONE. For the effects of the AUDIT value, see “Chapter 13. Protecting data sets” on page 215.
- Add or drop a parent key or a foreign key; see “Altering a table for referential integrity” on page 61.
- Change the VALIDPROC clause; see “Altering the assignment of a validation routine” on page 63.
- Change the DATA CAPTURE clause; see “Altering a table for capture of changed data” on page 64.
- Add or drop a table check constraint; see “Adding or dropping table check constraints” on page 63.
- Add or drop the restriction on dropping the table and the database and table space that contain the table; see *DB2 SQL Reference*.
- Redefine the attributes of an identity column; see “Redefining the attributes on an identity column” on page 68.
- Alter the length of a VARCHAR column using the SET DATA TYPE VARCHAR clause; see *DB2 SQL Reference*.

In addition, this section includes techniques for making the following changes:
“Changing an edit procedure or a field procedure” on page 64
“Altering the subtype of a string column” on page 65
“Moving a table to a table space of a different page size” on page 69

For other changes, you must drop and re-create the table as described in “Altering data types and deleting columns” on page 65.

## Adding a new column

When you use ALTER TABLE to add a new column, the new column becomes the rightmost column of the table. The physical records are not actually changed until values are inserted in the new column. Plans and packages are not invalidated unless the new column is a TIME, TIMESTAMP, or DATE. However, to use the new column in a program, you need to modify and recompile the program and bind the plan or package again. You might also need to modify any program containing a static SQL statement SELECT *, which will return the new column after the plan or package is rebound. You must also modify any INSERT statement not containing a column list.

Access time to the table is not affected immediately, unless the record was previously fixed length. If the record was fixed length, the addition of a new column causes DB2 to treat the record as variable length and then access time is affected immediately. To change the records to fixed length, follow these steps:

1. Run REORG with COPY on the table space, using the inline copy.
2. Run the MODIFY utility with the DELETE option to delete records of all image copies that were made before the REORG you ran in step 1.
3. Create a unique index if you add a column that specifies PRIMARY KEY.

Inserting values in the new column might also degrade performance by forcing rows onto another physical page. You can avoid this situation by creating the table space with enough free space to accommodate normal expansion. If you already have this problem, run REORG on the table space to fix it.

You can define the new column as NOT NULL by using the DEFAULT clause unless the column has a ROWID data type or is an identity column. If the column has a ROWID data type or is an identity column, you must specify NOT NULL without the DEFAULT clause.You can let DB2 choose the default value, or you can specify a constant or the value of the CURRENT SQLID or USER special register as the value to be used as the default. When you retrieve an existing row from the table, a default value is provided for the new column. Except in the following cases, the value for retrieval is the same as the value for insert:

- For columns of data type DATE, TIME, and TIMESTAMP, the retrieval defaults are:

| **Data Type** | **Default for Retrieval** |
|---|---|
| **DATE** | 0001-01-01 |
| **TIME** | 00.00.00 |
| **TIMESTAMP** | 0001-01-01-00.00.00.000000 |

- For DEFAULT USER and DEFAULT CURRENT SQLID, the value retrieved for rows that existed before the column was added is the value of the special register when the column was added.

If the new column is a ROWID column, DB2 returns the same, unique row ID value for a row each time you access that row. Reorganizing a table space does not affect the values on a ROWID column. You cannot use the DEFAULT clause for ROWID columns.

If the new column is an identity column (a numeric column that is defined with the AS IDENTITY clause), DB2 places the table space in REORG-pending (REORP) status, and access to the table space is restricted until the table space is reorganized. When the REORG utility is run, DB2

- Generates a unique value for the identity column of each existing row
- Physically stores these values in the database
- Removes the REORP status

You cannot use the DEFAULT clause for identity columns. For more information about identity columns, see *DB2 SQL Reference*.

If the new column is a short string column, you can specify a field procedure for it; see “Field procedures” on page 934. If you do specify a field procedure, you cannot also specify NOT NULL.

The following example adds a new column to the table DSN8710.DEPT, which contains a location code for the department. The column name is LOCNCODE, and its data type is CHAR (4).

```
ALTER TABLE DSN8710.DEPT
  ADD LOCNCODE CHAR (4);
```

## Altering a table for referential integrity

You can alter a table to add, change, or remove referential constraints. If you plan to let DB2 enforce referential integrity in a set of tables, then see Part 2 of *DB2 Application Programming and SQL Guide* for a description of the requirements for referential constraints. This section discusses these topics:

- "Adding referential constraints to existing tables"
- "Adding parent keys and foreign keys" on page 62
- "Dropping parent keys and foreign keys" on page 62
- "Adding or dropping table check constraints" on page 63

### Adding referential constraints to existing tables

Assume that the tables in the sample application already exist, have the appropriate column definitions, and are already populated. You want to define relationships among them by adding primary and foreign keys with the ALTER TABLE statement. The rules for these relationships are as follows:

- An existing table must have a unique index on its primary key columns before you can add the primary key. The index becomes the primary index.
- The parent key of the parent table must be added before the corresponding foreign key of the dependent table.

You can build the same referential structure several different ways. The following sequence does not have the fewest number of possible operations, but it is perhaps the simplest to explain.

1. Create a unique index on the primary key columns for any table that does not already have one.
2. For each table, issue an ALTER TABLE statement to add its primary key.

   In the next steps, you issue an ALTER TABLE statement to add foreign keys for each table except the activity table. This leaves the table space in check-pending status, which you reset by running CHECK DATA with the DELETE(YES) option.

   CHECK DATA deletes are not bound by delete rules; they cascade to all descendents of a deleted row. This can be disastrous. For example, if you delete the row for department A00 from the department table, the delete might propagate through most of the referential structure. The following steps prevent deletion from more than one table at a time.
3. Add the foreign keys for the department table and run CHECK DATA DELETE(YES) on its table space. Correct any rows in the exception table, and use INSERT to replace them in the department table. This table is now consistent with existing data.
4. Drop the foreign key on MGRNO in the department table. This "disconnects" it from the employee table, without changing its data.
5. Add the foreign key to the employee table, run CHECK DATA again, and correct any errors. If errors are reported, be particularly careful not to make any row inconsistent with the department table when you make corrections.
6. Again add the foreign key on MGRNO to the department table. This again leaves the table space in check pending status, so run CHECK DATA. If you

have not changed the data since the previous check, you can use DELETE(YES) with no fear of cascading deletions.

7. For each of the following tables, in the order shown, add its foreign keys, run CHECK DATA DELETE(YES), and correct any rows in error:
   a. Project table
   b. Project activity table
   c. Employee to project activity table

## Adding parent keys and foreign keys

You can add parent keys, both primary and unique, and foreign keys to an existing table.

To add a primary key to an existing table, use the PRIMARY KEY clause in an ALTER TABLE statement. For example, if the department table and its index XDEPT1 already exist, create its primary key by issuing:

```
ALTER TABLE DSN8710.DEPT
   ADD PRIMARY KEY (DEPTNO);
```

To add a unique key to an existing table, use the UNIQUE clause of the ALTER TABLE statement. For example, if the department table has a unique index defined on column DEPTNAME, you can add a unique key constraint, KEY_DEPTNAME consisting of column DEPTNAME by issuing:

```
ALTER TABLE DSN8710.DEPT
    ADD CONSTRAINT KEY_DEPTNAME UNIQUE (DEPTNAME);
```

Adding a parent key or a foreign key to an existing table has the following restrictions and implications:

- If you add a primary key, the table must already have a unique index on the key columns. The index that was most recently created on the key columns becomes the primary index. Because of the unique index, no duplicate values of the key exist in the table, therefore you do not need to check the validity of the data.
- If you add a unique key, the table must already have a unique index with a key that is identical to the unique key. DB2 arbitrarily chooses a unique index on the key columns to enforce the unique key. Because of the unique index, no duplicate values of the key exist in the table, therefore you do not need to check the validity of the data.
- You can use only one FOREIGN KEY clause in each ALTER TABLE statement; if you want to add two foreign keys to a table, you must execute two ALTER TABLE statements.
- If you add a foreign key, the parent key and unique index of the parent table must already exist. Adding the foreign key requires the ALTER privilege on the dependent table and either the ALTER or REFERENCES privilege on the parent table.
- Adding a foreign key establishes a relationship, with the many implications described in Part 2 of *DB2 Application Programming and SQL Guide*. DB2 does not validate the data. Instead, if the table is populated (or, in the case of a nonsegmented table space, if the table space has ever been populated), the table space containing the table is placed in check-pending status, just as if it had been loaded with ENFORCE NO. In this case, you need to execute CHECK DATA to clear the check-pending status.

## Dropping parent keys and foreign keys

You can drop parent keys, both primary and unique, and foreign keys from an existing table. Before you drop a foreign key or a parent key, consider carefully the effects on your application programs. The primary key of a table is intended to serve as a permanent, unique identifier of the occurrences of the entities it describes. Application programs often depend on that identifier. The foreign key defines a referential relationship and a delete rule. Without the key, your application programs must enforce the constraints.

When you drop a foreign key using the DROP FOREIGN KEY clause of the ALTER TABLE statement, DB2 drops the corresponding referential relationships. You must have the ALTER privilege on the dependent table and either the ALTER or REFERENCES privilege on the parent table. If the referential constraint references a unique key that has been created implicitly, and no other relationships are dependent on that unique key, the implicit unique key is also dropped.

When you drop a unique key using the DROP UNIQUE clause of the ALTER TABLE statement, DB2 drops all the referential relationships in which the unique key is a parent key; you must have the ALTER privilege on any dependent tables. As a result, the dependent tables no longer have foreign keys, and the table's unique index that enforced the unique key no longer indicates that it enforces a unique key although it is still a unique index.

When you drop a primary key using the DROP PRIMARY KEY clause of the ALTER TABLE statement, DB2 drops all the referential relationships in which the primary key is a parent key; you must have the ALTER privilege on any dependent tables. The dependent tables no longer have foreign keys; the table's primary index is no longer primary, but it is still a unique index.

### Adding or dropping table check constraints

You can define a check constraint on a table by using the ADD CHECK clause of the ALTER TABLE statement. If the table is empty, the check constraint is added to the description of the table.

If the table is not empty, what happens when you define the check constraint depends on the value of the CURRENT RULES special register, which can be either STD or DB2.

- If the value is STD, the check constraint is enforced immediately when it is defined. If a row does not conform, the table check constraint is not added to the table and an error occurs.
- If the value is DB2, the check constraint is added to the table description but its enforcement is deferred. Because some rows in the table might violate the check constraint, the table is placed in check-pending status.

The ALTER TABLE statement that is used to define a check constraint always fails if the table space or partition that contains the table is in a check-pending status, the CURRENT RULES special register value is STD, and the table is not empty.

To remove a check constraint from a table, use the DROP CONSTRAINT or DROP CHECK clauses of the ALTER TABLE statement. You must not use DROP CONSTRAINT on the same ALTER TABLE statement as DROP FOREIGN KEY, DROP CHECK, DROP PRIMARY KEY, or DROP UNIQUE.

## Altering the assignment of a validation routine

If you have a validation exit routine associated with a table, you can use the ALTER TABLE statement to make the following changes:

- Disassociate the validation routine from the table using the VALIDPROC NULL clause. The routine is no longer given control when DB2 accesses the table. For example:

```
ALTER TABLE DSN8710.EMP
  VALIDPROC NULL;
```

- Assign a new validation routine to the table using the VALIDPROC clause. (Only one validation routine can be connected to a table at a time; so if a validation routine already exists, DB2 disconnects the old one and connects the new routine.) Rows that existed before the connection of a new validation routine are not validated. In this example, the previous validation routine is disconnected and a new routine is connected with the program name EMPLNEWE:

```
ALTER TABLE DSN8710.EMP
  VALIDPROC EMPLNEWE;
```

### Checking rows of a table with a new validation routine

To ensure that the rows of a table conform to a new validation routine, you must run the validation routine against the old rows. One way to accomplish this is to use the REORG and LOAD utilities as shown in the following steps:

1. Use REORG to reorganize the table space that contains the table with the new validation routine. Specify UNLOAD ONLY, as in this example:

   ```
   REORG TABLESPACE DSN8D71A.DSN8S71E
     UNLOAD ONLY
   ```

   This step creates a data set that is used as input to the LOAD utility.
2. Run LOAD with the REPLACE option, and specify a discard data set to hold any invalid records. For example:

   ```
   LOAD INTO TABLE DSN8710.EMP
      REPLACE
      FORMAT UNLOAD
      DISCARDDN SYSDISC
   ```

The EMPLNEWE validation routine validates all rows after the LOAD step has completed. DB2 copies any invalid rows into the SYSDISC data set.

## Altering a table for capture of changed data

You can use DATA CAPTURE CHANGES on the ALTER TABLE statement to have data changes to that table written to the log in an expanded format. You can then retrieve the log by using a program such as the log apply feature of the Remote Recovery Data Facility (RRDF) program offering, or DB2 DataPropagator.

LOB values are not available for DATA CAPTURE CHANGES. To return a table back to normal logging, use DATA CAPTURE NONE.

## Changing an edit procedure or a field procedure

You cannot use ALTER TABLE to change the assignment of an edit procedure or a field procedure. However, with the assistance of DB2 utilities, you can change an existing edit procedure or field procedure.

To change an edit procedure or a field procedure for a table space in which the maximum record length is less than 32 KB, use the following procedure:

1. Run the UNLOAD utility or run the REORG utility with the UNLOAD EXTERNAL option to unload the data decoded by the existing edit procedure or field procedure.

   These utilities generate a LOAD statement in the data set (specified by the PUNCHDDN option) that you can use to reload the data into the original table space.

If you are using the same edit procedure or field procedure for many tables, unload the data from all the table spaces that have tables that use the procedure.

2. Modify the code for the edit procedure or the field procedure.
3. After the unload operation is completed, stop DB2.
4. Link-edit the new procedure, using the same name as the old procedure.
5. Start DB2.
6. Use the LOAD utility to reload the data. LOAD then uses the new edit procedure or field procedure to encode the data.

To change an edit procedure or a field procedure for a table space in which the maximum record length is greater than 32 KB, use the DSNTIAUL sample program to unload the data.

## Altering the subtype of a string column

If you add a column with a string data type, you can specify its subtype in the ALTER TABLE statement. Subtypes are valid for string columns of data types CHAR and VARCHAR. SBCS and MIXED are valid subtypes for CLOB data.

You can also change the subtype of an existing string column, but not by using ALTER TABLE. The operation involves updating the FOREIGN KEY column of the SYSIBM.SYSCOLUMNS catalog table and requires the SYSADM authority, SYSCTRL authority, or DBADM authority for the catalog database. The interpretation of the FOREIGNKEY column depends on whether the MIXED DATA install option is YES or NO.

If the MIXED DATA install option on installation panel DSNTIPF is YES, use one of the following values in the column:
**B** for bit data
**S** for SBCS data
**Any other value** for MIXED data

If the MIXED DATA install option is NO, use one of the following values in the column:
**B** for bit data
**Any other value** for SBCS data

Entering an M in the column when the MIXED DATA install option is NO specifies SBCS data, not MIXED data.

## Altering data types and deleting columns

Some changes to a table cannot be made with an ALTER TABLE statement. For example, an original specification of CHAR (20) must be lengthened to CHAR (25), a column defined as SMALLINT must be redefined as INTEGER, a column defined with NOT NULL must allow null values, or the attributes of an identity column must be redefined..

To make such changes, you need to perform the following steps:

1. Unload the table.
2. Drop the table.

*Be very careful about dropping a table*—in most cases, recovering a dropped table is nearly impossible. If you decide to drop a table, remember that such changes might invalidate a plan or a package, as described in “Dropping and re-creating DB2 objects” on page 55.

You must alter tables that have been created with RESTRICT ON DROP to remove the restriction before you can drop them.

3. Commit the changes.
4. Re-create the table.

   If the table has an identity column:

   - Choose carefully the new value for the START WITH attribute of the CREATE TABLE statement if you want the first generated column value to resume in sequence after the last generated column value of the table that was saved by the unload in step 1.
   - Define the table as GENERATED BY DEFAULT so that the previously generated identity values are reloaded.
5. Reload the table.

## Implications of dropping a table

The DROP TABLE statement deletes a table. For example, to drop the project table, execute:

```
DROP TABLE DSN8710.PROJ;
```

The statement deletes the row in the SYSIBM.SYSTABLES catalog table that contains information about DSN8710.PROJ. It *also* drops any other objects that depend on the project table. As a result:

- The column names of the table are dropped from SYSIBM.SYSCOLUMNS.
- If the dropped table has an identity column, all information regarding the identity column is removed from SYSIBM.SYSSEQUENCES.
- If triggers are defined on the table, they are dropped, and the corresponding rows are removed from SYSIBM.SYSTRIGGERS and SYSIBM.SYSPACKAGES.
- Any views based on the table are dropped.
- Application plans or packages that involve the use of the table are invalidated.
- Synonyms for the table are dropped from SYSIBM.SYSSYNONYMS.
- Indexes created on any columns of the table are dropped.
- Referential constraints that involve the table are dropped. In this case, the project table is no longer a dependent of the department and employee tables, nor is it a parent of the project activity table.
- Authorization information that is kept in the DB2 catalog authorization tables is updated to reflect the dropping of the table. Users who were previously authorized to use the table, or views on it, no longer have those privileges, because catalog rows are deleted.
- Access path statistics and space statistics for the table are deleted from the catalog.
- The storage space of the dropped table might be reclaimed.

  If the table space containing the table is:

  – Implicitly created (using CREATE TABLE without the TABLESPACE clause), the table space is also dropped. If the data sets are in a storage group, dropping the table space reclaims the space. For user-managed data sets, you must reclaim the space yourself.

  – Partitioned, or contains only the one table, you can drop the table space.

  – Segmented, DB2 reclaims the space.

- Simple, and contains other tables, you must run the REORG utility to reclaim the space.

- If the table contains a LOB column, the auxiliary table and the index on the auxiliary table are dropped. The LOB table space is dropped if it was created with SQLRULES(STD). See *DB2 SQL Reference*for details.

If a table has a partitioning index, you must drop the table space or use LOAD REPLACE when loading the redefined table. If the CREATE TABLE creates a table space implicitly, commit the DROP statement before re-creating a table by the same name. You must also commit the DROP statement before you create any new indexes with the same name as the original indexes.

## Check objects that depend on the table

Before dropping a table, check to see what other objects are dependent on it. The SYSIBM.SYSVIEWDEP, SYSIBM.SYSPLANDEP, and SYSIBM.SYSPACKDEP tables tell what views, application plans, and packages are dependent on different DB2 objects. This example lists the views, with their creators, that are affected if you drop the project table.

```
SELECT DNAME, DCREATOR
  FROM SYSIBM.SYSVIEWDEP
  WHERE BNAME = 'PROJ'
  AND BCREATOR = 'DSN8710'
  AND BTYPE = 'T';
```

The next example lists the packages, identified by the package name, collection ID, and consistency token (in hexadecimal representation), that are affected if you drop the project table.

```
SELECT DNAME, DCOLLID, HEX(DCONTOKEN)
  FROM SYSIBM.SYSPACKDEP
  WHERE BNAME = 'PROJ'
  AND BQUALIFIER = 'DSN8710'
  AND BTYPE = 'T';
```

This example lists the plans, identified by plan name, that are affected if you drop the project table.

```
SELECT DNAME
  FROM SYSIBM.SYSPLANDEP
  WHERE BNAME = 'PROJ'
  AND BCREATOR = 'DSN8710'
  AND BTYPE = 'T';
```

The SYSIBM.SYSINDEXES table tells you what indexes currently exist on a table. From the SYSIBM.SYSTABAUTH table, you can determine which users are authorized to use the table.

## Re-creating a table

To re-create a DB2 table to increase the length attribute of a string column or the precision of a numeric column, follow these steps:

1. If you do not have the original CREATE TABLE statement and all authorization statements for the table (call it T1), query the catalog to determine its description, the description of all indexes and views on it, and all users with privileges on it.
2. Create a new table (call it T2) with the desired attributes.
3. Execute the following INSERT statement:

   ```
   INSERT INTO T2
     SELECT * FROM T1;
   ```

This copies the contents of T1 into T2.

4. Execute the statement DROP TABLE T1. If T1 is the only table in an explicitly created table space, and you do not mind losing the compression dictionary, if one exists, drop the table space instead, so that the space is reclaimed.
5. Commit the DROP statement.
6. Use the statement RENAME TABLE to rename table T2 to T1.
7. Run the REORG utility on the table space that contains table T1.
8. Notify users to re-create any synonyms, indexes, views, and authorizations they had on T1.

If you want to change a data type from string to numeric or from numeric to string (for example, INTEGER to CHAR or CHAR to INTEGER), use the CHAR and DECIMAL scalar functions in the SELECT statement to do the conversion.Another alternative is to:

1. Use UNLOAD or REORG UNLOAD EXTERNAL (if the data to unload in less than 32 KB) to save the data in a sequential file, and then
2. Use the LOAD utility to repopulate the table after re-creating it. When you reload the table, make sure you edit the LOAD statement to match the new column definition.

This method is particularly appealing when you are trying to re-create a large table.

## Redefining the attributes on an identity column

At some point, you might need to change the attributes of an identity column. For example, you might want to allow or disallow identity column values to wrap, or change values of other attributes. Changing the attributes of an identity column, like changing other data types, requires that you drop and then recreate the table (see "Altering data types and deleting columns" on page 65 for special considerations for altering identity columns).

If you want identity column values to wrap, specify the CYCLE attribute on the SQL statements CREATE TABLE, DECLARE GLOBAL TEMPORARY TABLE, or ALTER TABLE. The CYCLE attribute lets an identity column continue to generate values after it reaches the minimum or maximum value of the cycle sequence. When wrapping is in effect, duplicate column values are implicitly allowed (even when the column is GENERATED ALWAYS) unless a unique index is defined on the column. *To ensure unique values for an identity column, you must define a unique index on that column.*

If you do not want identity column values to wrap, specify the NO CYCLE attribute (the default if both CYCLE and NO CYCLE are not specified). If the NO CYCLE attribute is in effect and you run out of values, then you must:

1. Unload the table.
2. Drop the table.
3. Re-create the table, specifying:
   - New values for attributes (such as data type, MAXVALUE, and MINVALUE) that allow a larger range, if possible
   - GENERATED BY DEFAULT so that you can reload existing values
   - An appropriate START WITH value that allows the sequence to continue from where it ended in the unloaded table
4. Reload the table.

The attributes MINVALUE and MAXVALUE on the AS IDENTITY clause let you specify the minimum and maximum values that are generated for an identity column. See Chapter 5 of *DB2 SQL Reference* for more information about identity column attributes.

## Moving a table to a table space of a different page size

You cannot alter a table to use a different page size. However, you can move a table to a table space of a different page size:

1. Unload the table using UNLOAD FROM TABLE or REORG UNLOAD EXTERNAL FROM TABLE.
2. Use CREATE TABLE LIKE on the table to re-create it in the table space of the new page size.
3. Use DB2 Control Center, DB2 Administration Tool, or catalog queries to determine the dependent objects: views, authorization, plans, packages, synonyms, triggers, referential integrity, and indexes.
4. Drop the original table.
5. Rename the new table to the name of the old table using RENAME TABLE.
6. Re-create all dependent objects.
7. Rebind plans and packages.
8. Reload the table using data from the SYSRECnn data set and the control statements from the SYSPUNCH data set, which was created when the table was unloaded.

# Altering indexes

You can use the ALTER INDEX statement to change the description of an index or to rebalance data among partitions in partitioned table spaces. The statement can be embedded in an application program or issued interactively. For details on the ALTER INDEX statement, see Chapter 5 of *DB2 SQL Reference*.

## Changing the description of an index

You can change most of the index clauses, including: BUFFERPOOL, CLOSE, COPY, PIECESIZE, and those clauses that are associated with storage space, free space, and group bufferpool caching. To change any other clause of the index definition, you must drop the index, commit, and redefine it. Dropping an index does not cause DB2 to drop any other objects. As described in “Dropping and re-creating DB2 objects” on page 55, the consequence of dropping indexes is that DB2 invalidates application plans and packages that use the index and automatically rebinds them when they are next used.

If you want to drop a unique index, you must take additional steps before running a DROP INDEX statement. Any primary key, unique key, or referential constraints associated with the unique index must be dropped before you drop the unique index. However, you can drop a unique index for a unique key without dropping the unique constraint if the unique key was created before Version 7.

You must commit the DROP INDEX statement before you create any new table spaces or indexes by the same name. If an index is dropped and then an application program using that index is run (and thereby automatically rebound), that application program does not use the old index. If, at a later time, the index is re-created, and the application program is not rebound, the application program cannot take advantage of the new index.

## Rebalancing data in partitioned table spaces

When data becomes out of balance in partitioned table spaces, performance can decrease. It is possible to improve performance by rebalancing the partitions to redistribute the data. You can use the ALTER INDEX statement and a REORG job to shift data among the affected partitions. The result is that data is is balanced according to your specifications.

You can rebalance data by changing the limit key values of all or most of the partitions. The limit key is the highest value of the index key for a partition. You roll the changes through the partitions one or more at a time, making relatively small parts of the data unavailable at any given time. For more information about rebalancing data using the ALTER INDEX statement, see *An Introduction to DB2 for OS/390*.

## Altering views

In many cases, changing user requirements can be satisfied by modifying an existing view. But no ALTER VIEW statement exists; the only way to change a view is by dropping the view, committing the drop, and re-creating the view. When you drop a view, DB2 also drops the dependent views.

When you drop a view, DB2 invalidates application plans and packages that are dependent on the view and revokes the privileges of users who are authorized to use it. DB2 attempts to rebind the package or plan the next time it is executed, and you receive an error if you do not re-create the view.

To tell how much rebinding and reauthorizing is needed if you drop a view, check these catalog tables:

*Table 12. Catalog tables to check after dropping a view*

| Catalog table | What to check |
|---|---|
| SYSIBM.SYSPLANDEP | Application plans dependent on the view |
| SYSIBM.SYSPACKDEP | Packages dependent on the view |
| SYSIBM.SYSVIEWDEP | Views dependent on the view |
| SYSIBM.SYSTABAUTH | Users authorized to use the view |

For more information about defining and dropping views, see Chapter 5 of *DB2 SQL Reference*.

## Altering stored procedures and user-defined functions

You modify stored procedures and user-defined functions with the ALTER PROCEDURE and ALTER FUNCTION statements. Details for these statements are in *DB2 Application Programming and SQL Guide*.

### Altering stored procedures

The ALTER PROCEDURE statement updates the description of a stored procedure. This example changes SYSPROC.MYPROC to run only in the WLM environment PARTSEC:

```
ALTER PROCEDURE SYSPROC.MYPROC
   WLM ENVIRONMENT PARTSEC;
```

If SYSPROC.MYPROC is defined with SECURITY DEFINER, the external security environment for the stored procedure uses the authorization ID of the owner of the stored procedure. This example changes the procedure to use the authorization ID of the person running it:

```
ALTER PROCEDURE SYSPROC.MYPROC
   SECURITY USER;
```

## Altering user-defined functions

The ALTER FUNCTION statement updates the description of user-defined functions. Changes take effect immediately.

In this example, two functions named CENTER exist in the PELLOW schema. The first function has two input parameters with INTEGER and FLOAT data types, respectively. The specific name for the first function is FOCUS1. The second function has three parameters with CHAR(25), DEC(5,2), and INTEGER data types.

Using the specific name to identify the function, change the WLM environment in which the first function runs from WLMENVNAME1 to WLMENVNAME2:

```
ALTER SPECIFIC FUNCTION PELLOW.FOCUS1
   WLM ENVIRONMENT WLMENVNAME2;
```

This example changes the second function when any arguments are null:

```
ALTER FUNCTION PELLOW.CENTER (CHAR(25), DEC(5,2), INTEGER)
   RETURNS ON NULL CALL;
```

# Changing the high-level qualifier for DB2 data sets

The high-level qualifier for DB2 data sets is the catalog name of the integrated catalog facility, which is commonly called the “user catalog”. You cannot change this qualifier for DB2 data sets using the DB2 installation or migration update process. This section describes other ways to change this qualifier for both system data sets and user data sets.

These procedures do not actually move or copy data. For information about moving data, see “Moving DB2 data” on page 78.

Changing the high-level qualifier for DB2 data sets is a complex task; so, you should have both experience with DB2 and with managing user catalogs. The following tasks are described:

- “Define a new integrated catalog alias” on page 72
- “Change the qualifier for system data sets” on page 72, which includes the DB2 catalog, directory, active and archive logs, and the BSDS
- “Change qualifiers for other databases and user data sets” on page 75, which includes the work file database (DSNDB07), the default database (DSNDB04), and other DB2 databases and user databases

To concentrate on DB2-related issues, this procedure assumes that the catalog alias resides in the *same* user catalog as the one that is currently used. If the new catalog alias resides in a *different* user catalog, refer to *DFSMS/MVS: Access Method Services for the Integrated Catalog* for information about planning such a move.

If the data sets are managed by the Storage Management Subsystem (SMS), make sure that automatic class selection routines are in place for the new data set name.

## Define a new integrated catalog alias

This step can be done at any time before changing the high-level qualifier for system or user data sets.

Set up the new high-level qualifier as an alias to a current integrated catalog, using the following command:

```
DEFINE ALIAS (NAME (newcat) RELATE (usercat) CATALOG (master-cat))
```

See *DFSMS/MVS: Access Method Services for the Integrated Catalog* for more information.

## Change the qualifier for system data sets

In this task, you stop DB2, change the high-level qualifier in the system parameter load module (possibly DSNZPARM), and establish a new xxxxMSTR cataloged procedure before restarting DB2. These steps must be done in sequence.

### Step 1: Change the load module to reflect the new qualifier

To change the system parameter load module to specify the new qualifier for new archive data sets and the DB2 catalog and directory data sets, you must use the installation process.

1. Run the installation CLIST, and specify INSTALL TYPE=INSTALL and DATA SHARING FUNCTION=NONE.
2. Enter new values for the fields shown in Table 13.

*Table 13. CLIST panels and fields to change to reflect new qualifier*

| Panel name | Field name | Comments |
|---|---|---|
| DSNTIPA1 | INSTALL TYPE | Specify INSTALL. Do *not* specify a new default prefix for the input data sets listed on this panel. |
| DSNTIPA1 | OUTPUT MEMBER NAME | |
| DSNTIPA2 | CATALOG ALIAS | |
| DSNTIPH | COPY 1 NAME and COPY 2 NAME | These are the bootstrap data set names. |
| DSNTIPH | COPY 1 PREFIX and COPY 2 PREFIX | These fields appear for both active and archive log prefixes. |
| DSNTIPT | SAMPLE LIBRARY | This field allows you to specify a field name for edited output of the installation CLIST. Avoid overlaying existing data sets by changing the middle node, NEW, to something else. The only members you use in this procedure are xxxxMSTR and DSNTIJUZ in the sample library. |
| DSNTIPO | PARAMETER MODULE | Change this value only if you want to preserve the existing member through the CLIST. |

   The output from the CLIST is a new set of tailored JCL with new cataloged procedures and a DSNTIJUZ job, which produces a new member.

3. Run DSNTIJUZ.

Unless you have specified a new name for the load module, make sure the output load module does not go to the SDSNEXIT or SDSNLOAD library used by the active DB2 subsystem.

DSNTIJUZ also places any archive log data sets into the BSDS and creates a new DSNHDECP member. You do not need to run these steps, because they are unnecessary for changing the high-level qualifier.

## Step 2: Stop DB2 with no outstanding activity

In this step, make sure the subsystem does not have any outstanding activity, such as outstanding units of recovery or pending writes. This ensures that DB2 does not need to access the data sets on restart through the log, which contains the *old* data set qualifiers.

1. Enter the following command:

   ```
   -STOP DB2 MODE(QUIESCE)
   ```

   This command allows DB2 to complete processing currently executing programs.
2. Enter the following command:

   ```
   -START DB2 ACCESS(MAINT)
   ```
3. Use the following commands to make sure the subsystem is in a consistent state. See Chapter 2 of *DB2 Command Reference* and “Part 4. Operation and recovery” on page 241 for more information about these commands.

   ```
   -DISPLAY THREAD(*) TYPE(*)
   -DISPLAY UTILITY (*)
   -TERM UTILITY(*)
   -DISPLAY DATABASE(*) RESTRICT
   -DISPLAY DATABASE(*) SPACENAM(*) RESTRICT
   -RECOVER INDOUBT
   ```

   Correct any problems before continuing.
4. Stop DB2, using the following command:

   ```
   -STOP DB2 MODE(QUIESCE)
   ```
5. Run the print log map utility (DSNJU004) to identify the current active log data set and the last checkpoint RBA. For information about the print log map utility, see Part 3 of *DB2 Utility Guide and Reference*.
6. Run DSN1LOGP with the SUMMARY (YES) option, using the last checkpoint RBA from the output of the print log map utility you ran in the previous step. For information about DSN1LOGP, see Part 3 of *DB2 Utility Guide and Reference*.

   The report headed DSN1157I RESTART SUMMARY identifies active units of recovery or pending writes. If either situation exists, do not attempt to continue. Start DB2 with ACCESS(MAINT), use the necessary commands to correct the problem, and repeat steps 4 through 6 until all activity is complete.

## Step 3: Rename system data sets with the new qualifier

All of the following steps assume that the new qualifier and the old qualifier reside in the same user catalog. Access method services does not allow ALTER where the new name does not match the existing catalog structure for an SMS-managed VSAM data set. If the data set is not managed by SMS, the rename succeeds, but DB2 cannot allocate it as described in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

DB2 table spaces are defined as linear data sets with DSNDBC as the second node of the name for the cluster and DSNDBD for the data component (as described in “Requirements for your own data sets” on page 34). The examples shown here assume the normal defaults for DB2 and VSAM data set names. Use access method services statements with a generic name (*) to simplify the process. Access method services allows only one generic name per data set name string.

1. Using IDCAMS, change the names of the catalog and directory table spaces. Also, be sure to specify the instance qualifier of your data set, *y*, which can be either I or J:

   ```
   ALTER oldcat.DSNDBC.DSNDB01.*.y0001.A001  -
      NEWNAME (newcat.DSNDBC.DSNDB01.*.y0001.A001)
   ALTER oldcat.DSNDBD.DSNDB01.*.y0001.A001  -
      NEWNAME (newcat.DSNDBD.DSNDB01.*.y0001.A001)
   ALTER oldcat.DSNDBC.DSNDB06.*.y0001.A001  -
      NEWNAME (newcat.DSNDBC.DSNDB06.*.y0001.A001)
   ALTER oldcat.DSNDBD.DSNDB06.*.y0001.A001  -
      NEWNAME (newcat.DSNDBD.DSNDB06.*.y0001.A001)
   ```

2. Using IDCAMS, change the active log names. Active log data sets are named *oldcat*.LOGCOPY1.COPY01 for the cluster component and *oldcat*.LOGCOPY1.COPY01.DATA for the data component.

   ```
   ALTER oldcat.LOGCOPY1.*  -
      NEWNAME (newcat.LOGCOPY1.*)
   ALTER oldcat.LOGCOPY1.*.DATA  -
      NEWNAME (newcat.LOGCOPY1.*.DATA)
   ALTER oldcat.LOGCOPY2.*  -
      NEWNAME (newcat.LOGCOPY2.*)
   ALTER oldcat.LOGCOPY2.*.DATA  -
      NEWNAME (newcat.LOGCOPY2.*.DATA)
   ```

3. Using IDCAMS, change the BSDS names.

   ```
   ALTER oldcat.BSDS01  -
      NEWNAME (newcat.BSDS01)
   ALTER oldcat.BSDS01.*  -
      NEWNAME (newcat.BSDS01.*)
   ALTER oldcat.BSDS02  -
      NEWNAME (newcat.BSDS02)
   ALTER oldcat.BSDS02.*  -
      NEWNAME (newcat.BSDS02.*)
   ```

### Step 4: Update the BSDS with the new qualifier

Update the first BSDS with the new alias and correct data set names for the active logs. This procedure does not attempt to change the names of existing archive log data sets. If these catalog entries or data sets will not be available in the future, copy all the table spaces in the DB2 subsystem to establish a new recovery point. You can optionally delete the entries from the BSDS. If you do not delete the entries, they will gradually be replaced by newer entries.

1. Run the change log inventory utility (DSNJU003).

   Use the *new* qualifier for the BSDS, because it has now been renamed. The following example illustrates the control statements required for three logs and dual copy is specified for the logs. This is only an example; the number of logs can vary and dual copy is an option. The starting and ending log RBAs are from the print log map report.

   ```
   NEWCAT VSAMCAT=newcat
   DELETE DSNAME=oldcat.LOGCOPY1.DS01
   DELETE DSNAME=oldcat.LOGCOPY1.DS02
   DELETE DSNAME=oldcat.LOGCOPY1.DS03
   DELETE DSNAME=oldcat.LOGCOPY2.DS01
   DELETE DSNAME=oldcat.LOGCOPY2.DS02
   DELETE DSNAME=oldcat.LOGCOPY2.DS03
   NEWLOG DSNAME=newcat.LOGCOPY1.DS01,COPY1,STARTRBA=strtrba,ENDRBA=endrba
   NEWLOG DSNAME=newcat.LOGCOPY1.DS02,COPY1,STARTRBA=strtrba,ENDRBA=endrba
   NEWLOG DSNAME=newcat.LOGCOPY1.DS03,COPY1,STARTRBA=strtrba,ENDRBA=endrba
   ```

```
NEWLOG DSNAME=newcat.LOGCOPY2.DS01,COPY2,STARTRBA=strtrba,ENDRBA=endrba
NEWLOG DSNAME=newcat.LOGCOPY2.DS02,COPY2,STARTRBA=strtrba,ENDRBA=endrba
NEWLOG DSNAME=newcat.LOGCOPY2.DS03,COPY2,STARTRBA=strtrba,ENDRBA=endrba
```

During startup, DB2 compares the *newcat* value with the value in the system parameter load module, and they must be the same.

2. Using the IDCAMS REPRO command, replace the contents of BSDS2 with the contents of BSDS01.
3. Run the print log map utility (DSNJU004) to verify your changes to the BSDS.
4. At a convenient time, change the DD statements for the BSDS in any of your off-line utilities to use the new qualifier.

### Step 5: Establish a new xxxxmstr cataloged procedure

Before you start DB2, follow these steps:

1. Update xxxxMSTR in SYS1.PROCLIB with the new BSDS data set names.
2. Copy the new system parameter load module to the active SDSNEXIT/SDSNLOAD library.

### Step 6: Start DB2 with the new xxxxmstr and load module

Use the START DB2 command with the new load module name as shown here:

```
-START DB2 PARM(new_name)
```

If you stopped DSNDB01 or DSNDB06 in “Step 2: Stop DB2 with no outstanding activity” on page 73, you must explicitly start them in this step.

## Change qualifiers for other databases and user data sets

This step changes qualifiers for DB2 databases other than the catalog and directory. DSNDB07 is a system database but contains no permanent data, and can be deleted and redefined with the new qualifier. If you are changing its qualifier, do that before the rest of the user databases.

Change the databases in the following list that apply to your environment.

DSNDB07 (work file database)
DSNDB04 (default database)
DSNDDF (communications database)
DSNRLST (resource limit facility database)
DSNRGFDB (the database for data definition control)
Any other application databases that use the old high-level qualifier

At this point, the DB2 catalog tables SYSSTOGROUP, SYSTABLEPART, and SYSINDEXPART contain information about the old integrated user catalog alias. To update those tables with the new alias, you must follow the procedures below. Until you do so, the underlying resources are not available. The following procedures are described separately.

- “Changing your work database to use the new high-level qualifier” on page 76
- “Changing user-managed objects to use the new qualifier” on page 76
- “Changing DB2-managed objects to use the new qualifier” on page 77

Table spaces and indexes that span more than one data set require special procedures. Partitioned table spaces can have different partitions allocated to different DB2 storage groups. Nonpartitioned table spaces or indexes only have the additional data sets to rename (those with the lowest level name of A002, A003, and so on).

## Changing your work database to use the new high-level qualifier

You can use one of two methods to change the high-level qualifier for your work database or possibly DSNDB07. Which method you use depends on if you have a new installation or a migrated installation.

***New installation:***

1. Reallocate the database using the installation job DSNTIJTM from *prefix*.SDSNSAMP
2. Modify your existing job. Change the job to remove the BIND step for DSNTIAD and rename the data set names in the DSNTTMP step to your new names, making sure you include your *current* allocations.

***Migrated installations:*** Migrated installations do not have a usable DSNTIJTM, because the IDCAMS allocation step is missing. For migrated installations, you must:

1. Stop the database, using the following command (for a database named DSNDB07):
   ```
   -STOP DATABASE (DSNDB07)
   ```
2. Drop the database, using the following SQL statement:
   ```
   DROP DATABASE DSNDB07;
   ```
3. Re-create the database, using the following SQL statement:
   ```
   CREATE DATABASE DSNDB07;
   ```
4. Define the clusters, using the following access method services commands. Also, be sure to specify the instance qualifier of your data set, *y*, which can be either I or J:
   ```
      ALTER oldcat.DSNDBC.DSNDB07.DSN4K01.y0001.A001
         NEWNAME newcat.DSNDBC.DSNDB07.DSN4K01.y0001.A001
      ALTER oldcat.DSNDBC.DSNDB07.DSN32K01.y0001.A001
         NEWNAME newcat.DSNDBC.DSNDB07.DSN32K01.y0001.A001
   ```

   Repeat the above statements (with the appropriate table space name) for as many table spaces as you use.
5. Create the table spaces in DSNDB07.
   ```
   CREATE TABLESPACE DSN4K01
     IN DSNDB07
     BUFFERPOOL BP0
     CLOSE NO
     USING VCAT DSNC710;

   CREATE TABLESPACE DSN32K01
     IN DSNDB07
     BUFFERPOOL BP32K
     CLOSE NO
     USING VCAT DSNC710;
   ```
6. Start the database, using the following command:
   ```
   -START DATABASE (DSNDB07)
   ```

## Changing user-managed objects to use the new qualifier

1. Stop the table spaces and index spaces, using the following command:
   ```
   -STOP DATABASE(dbname) SPACENAM(*)
   ```
2. Use the following SQL ALTER TABLESPACE and ALTER INDEX statements with the USING clause to specify the new qualifier:
   ```
   ALTER TABLESPACE dbname.tsname
     USING VCAT newcat;

   ALTER INDEX creator.index-name
     USING VCAT newcat;
   ```

Repeat for all the objects in the database.

3. Using IDCAMS, rename the data sets to the new qualifier. Also, be sure to specify the instance qualifier of your data set, *y*, which can be either I or J:

   ```
   ALTER oldcat.DSNDBC.dbname.*.y0001.A001  -
      NEWNAME newcat.DSNDBC.dbname.*.y0001.A001
   ALTER oldcat.DSNDBD.dbname.*.y0001.A001  -
      NEWNAME newcat.DSNDBD.dbname.*.y0001.A001
   ```

4. Start the table spaces and index spaces, using the following command:

   ```
   -START DATABASE(dbname) SPACENAM(*)
   ```

5. Verify the success of the procedure by entering the following command:

   ```
   -DISPLAY DATABASE(dbname)
   ```

6. Using SQL, verify that you can access the data.

Renaming the data sets can be done while DB2 is down. They are included here because the names must be generated for each database, table space, and index space that is to change.

## Changing DB2-managed objects to use the new qualifier

Use this procedure when you want to keep the existing DB2 storage group, changing only the high-level qualifier. If you want to move the data to a *new* DB2 storage group, see page 81.

1. Remove all table spaces and index spaces from the storage group by converting the data sets temporarily to user-managed data sets.
   a. Stop each database that has data sets you are going to convert, using the following command:

      ```
      -STOP DATABASE(dbname) SPACENAM(*)
      ```

   b. Convert to user-managed data sets with the USING VCAT clause of the SQL ALTER TABLESPACE and ALTER INDEX statements, as shown in the following statements. Use the new catalog name for VCAT.

      ```
      ALTER TABLESPACE dbname.tsname
          USING VCAT newcat;

      ALTER INDEX creator.index-name
          USING VCAT newcat;
      ```

2. Drop the storage group, using the following statement:

   ```
   DROP STOGROUP stogroup-name;
   ```

   The DROP succeeds only if all the objects that referenced this STOGROUP are dropped or converted to user-managed (USING VCAT clause).

3. Re-create the storage group using the correct volumes and the new alias, using the following statement:

   ```
   CREATE STOGROUP stogroup-name
      VOLUMES (VOL1,VOL2)
      VCAT newcat;
   ```

4. Using IDCAMS, rename the data sets for the index spaces and table spaces to use the new high-level qualifier. Also, be sure to specify the instance qualifier of your data set, *y*, which can be either I or J:

   ```
   ALTER oldcat.DSNDBC.dbname.*.y0001.A001  -
      NEWNAME newcat.DSNDBC.dbname.*.y0001.A001
   ALTER oldcat.DSNDBD.dbname.*.y0001.A001  -
      NEWNAME newcat.DSNDBD.dbname.*.y0001.A001
   ```

   If your table space or index space spans more than one data set, be sure to rename those data sets also.

5. Convert the data sets back to DB2-managed data sets by using the new DB2 storage group. Use the following SQL ALTER TABLESPACE and ALTER INDEX statements:

   ```
   ALTER TABLESPACE dbname.tsname
     USING STOGROUP stogroup-name
     PRIQTY priqty
     SECQTY secqty;
   ```

   ```
   ALTER INDEX creator.index-name
     USING STOGROUP stogroup-name
     PRIQTY priqty
     SECQTY secqty;
   ```

   If you specify USING STOGROUP without specifying the PRIQTY and SECQTY clauses, DB2 uses the default values. For more information about USING STOGROUP, see *DB2 SQL Reference*.
6. Start each database, using the following command:

   ```
   -START DATABASE(dbname) SPACENAM(*)
   ```
7. Verify the success of the procedure by entering the following command:

   ```
   -DISPLAY DATABASE(dbname)
   ```
8. Using SQL, verify that you can access the data.

## Moving DB2 data

This section discusses the following topics:

- “Tools for moving DB2 data” introduces some of the tools available to make moving DB2 data easier.
- “Moving a DB2 data set” on page 80 describes moving a data set from one volume to another.
- “Copying a relational database” on page 81 describes copying a user-managed relational database, with its object definitions and its data, from one DB2 subsystem to another, on the same or different MVS system.
- “Copying an entire DB2 subsystem” on page 81 describes copying a DB2 subsystem from one MVS system to another. Copying a subsystem includes these items:
  - All the user data and object definitions
  - The DB2 system data sets:
    - The log
    - The bootstrap data set
    - Image copy data sets
    - The DB2 catalog
    - The integrated catalog that records all the DB2 data sets

## Tools for moving DB2 data

**Important:** Before copying any DB2 data, resolve any data that is in an inconsistent state. Use the DISPLAY DATABASE command to determine whether any inconsistent state exists, and the RECOVER INDOUBT command or the RECOVER utility to resolve the inconsistency. The copying process generally loses all traces of an inconsistency except the problems that result.

Although DB2 data sets are created using VSAM access method services, they are specially formatted for DB2 and cannot be processed by services that use VSAM record processing. They can be processed by VSAM utilities that use control-interval (CI) processing and, if they are linear data sets (LDSs), also by utilities that recognize the LDS type.

Furthermore, copying the data might not be enough. Some operations require copying DB2 object definitions. And when copying from one subsystem to another, you must consider internal values that appear in the DB2 catalog and the log, for example, the DB2 object identifiers (OBIDs) and log relative byte addresses (RBAs).

Fortunately, several tools exist that simplify the operations:

- The REORG and LOAD utilities.

  Those can be used to move data sets from one disk device type to another within the same DB2 subsystem. For instructions on using LOAD and REORG, see Part 2 of *DB2 Utility Guide and Reference*.
- The COPY and RECOVER utilities. Using those utilities, you can recover an image copy of a DB2 table space or index space onto another disk device within the same subsystem. For instructions on using COPY and RECOVER, see Part 2 of *DB2 Utility Guide and Reference*.
- The UNLOAD or REORG UNLOAD EXTERNAL utility unloads a DB2 table into a sequential file and generates statements to allow the LOAD utility to load it elsewhere. For instructions on using UNLOAD or REORG UNLOAD EXTERNAL, see *DB2 Utility Guide and Reference*.
- The DSN1COPY utility. The utility copies the data set for a table space or index space to another data set. It can also translate the object identifiers and reset the log RBAs in the target data set. For instructions, see Part 3 of *DB2 Utility Guide and Reference*.

The following tools are not parts of DB2 but are separate licensed programs or program offerings:

- DB2 DataPropagator. This licensed program can extract data from DB2 tables, DL/I databases, VSAM files, and sequential files. For instructions, see “Loading data from DL/I” on page 54.
- DFSMS/MVS®, which contains the following functional components:
  - Data Set Services (DFSMSdss™)

    Use DFSMSdss to copy data between disk devices. For instructions, see *Data Facility Data Set Services: User's Guide and Reference*. You can use online panels to control this, through the Interactive Storage Management Facility (ISMF) that is available with DFSMS/MVS; for instructions, refer to *DFSMS/MVS: Storage Administration Reference for DFSMSdfp*.
  - Data Facility Product (DFSMSdfp)

    This is a prerequisite for DB2. You can use access method services EXPORT and IMPORT commands with DB2 data sets when control interval processing (CIMODE) is used. For instructions on EXPORT and IMPORT, see *DFSMS/MVS: Access Method Services for the Integrated Catalog*.
  - Hierarchical Storage Manager (DFSMShsm)

    With the MIGRATE, HMIGRATE, or HRECALL commands, which can specify specific data set names, you can move data sets from one disk device type to another within the same DB2 subsystem. Do not migrate the DB2 directory, DB2 catalog, and the work file database (DSNDB07). Do not migrate any data sets that are in use frequently, such as the bootstrap data set and the active log. With the MIGRATE VOLUME command, you can move an entire disk volume from one device type to another. The program can be controlled using online panels, through the Interactive Storage Management Facility (ISMF). For instructions, see *DFSMS/MVS: DFSMShsm Managing Your Own Data*.

The following table shows which tools are applicable to which operations:

*Table 14. Tools applicable to data-moving operations*

| **Tool** | **Moving a data set** | **Copying a data base** | **Copying an entire subsystem** |
|---|---|---|---|
| REORG and LOAD | Yes | Yes | — |
| COPY and RECOVER | Yes | — | — |
| DSNTIAUL | Yes | Yes | — |
| DSN1COPY | Yes | Yes | — |
| DataRefresher or DXT™ | Yes | Yes | — |
| DFSMSdss | Yes | — | Yes |
| DFSMSdfp | Yes | — | Yes |
| DFSMShsm | Yes | — | — |

Some of the listed tools rebuild the table space and index space data sets, and they therefore generally require longer to execute than the tools that merely copy them. The tools that rebuild are REORG and LOAD, RECOVER and REBUILD, DSNTIAUL, and DataRefresher. The tools that merely copy data sets are DSN1COPY, DFSMSdss, DFSMSdfp EXPORT and IMPORT, and DFSMShsm.

DSN1COPY is fairly efficient in use, but somewhat complex to set up. It requires a separate job step to allocate the target data sets, one job step for each data set to copy the data, and a step to delete or rename the source data sets. DFSMSdss, DFSMSdfp, and DFSMShsm all simplify the job setup significantly.

Although less efficient in execution, RECOVER is easy to set up if image copies and recover jobs already exist. You might only need to redefine the data sets involved and recover the objects as usual.

## Moving a DB2 data set

The movement DB2 data is accomplished by RECOVER, REORG, or DSN1COPY, or by the use of non-DB2 facilities, such as DFSMSdss. Both the DB2 utilities and the non-DB2 tools can be used while DB2 is running, but the space to be moved should be stopped to prevent users from accessing it.

The following procedures differ mainly in that the first one assumes you do not want to reorganize or recover the data. Generally, this means that the first procedure is faster. In all cases, make sure that there is enough space on the target volume to accommodate the data set.

**If you use storage groups**, then you can change the storage group definition to include the new volumes, as described in “Altering DB2 storage groups” on page 56.

***Moving data without REORG or RECOVER:***

1. Stop the database.
2. Move the data, using DSN1COPY or a non-DB2 facility.

3. Issue the ALTER INDEX or ALTER TABLESPACE statement to use the new integrated catalog facility catalog name or DB2 storage group name.
4. Start the database.

***Moving DB2-Managed data with REORG, RECOVER, or REBUILD:*** With this procedure you create a storage group (possibly using a new catalog alias) and move the data to that new storage group.

1. Create a new storage group using the correct volumes and the new alias, as shown in the following statement:

   ```
   CREATE STOGROUP stogroup-name
      VOLUMES (VOL1,VOL2)
      VCAT (newcat);
   ```

2. Prevent access to the data sets you are going to move, by entering the following command:

   ```
   -STOP DATABASE(dbname) SPACENAM(*)
   ```

3. Enter the ALTER TABLESPACE and ALTER INDEX SQL statements to use the new storage group name, as shown in the following statements:

   ```
   ALTER TABLESPACE dbname.tsname
     USING STOGROUP stogroup-name;
   ```

   ```
   ALTER INDEX creator.index-name
     USING STOGROUP stogroup-name;
   ```

4. Using IDCAMS, rename the data sets for the index spaces and table spaces to use the new high-level qualifier. Also, be sure to specify the instance qualifier of your data set, *y*, which can be either I or J. If you have run REORG with SHRLEVEL CHANGE or SHRLEVEL REFERENCE on any table spaces or index spaces, the fifth-level qualifier might be J0001.

   ```
      ALTER oldcat.DSNDBC.dbname.*.y0001.A001  -
         NEWNAME newcat.DSNDBC.dbname.*.y0001.A001
      ALTER oldcat.DSNDBD.dbname.*.y0001.A001  -
         NEWNAME newcat.DSNDBD.dbname.*.y0001.A001
   ```

5. Start the database for utility processing only, using the following command:

   ```
   -START DATABASE(dbname) SPACENAM(*) ACCESS(UT)
   ```

6. Use the REORG or the RECOVER utility on the table space or index space, or use the REBUILD utility on the index space.
7. Start the database, using the following command:

   ```
   -START DATABASE(dbname) SPACENAM(*)
   ```

## Copying a relational database

This operation involves not only copying data, but finding or generating, and executing, SQL statements to create storage groups, databases, table spaces, tables, indexes, views, synonyms, and aliases.

As with the other operations, DSN1COPY is likely to execute faster than the other applicable tools. It copies directly from one data set to another, while the other tools extract input for LOAD, which then loads table spaces and builds indexes. But again, DSN1COPY is more difficult to set up. In particular, you must know the internal DB2 object identifiers, which other tools translate automatically.

## Copying an entire DB2 subsystem

This operation involves copying an entire DB2 subsystem from one MVS system to another. (Although you can have two DB2 subsystems on the same MVS system, one cannot be a copy of the other.)

Only two of the tools listed are applicable: DFSMSdss DUMP and RESTORE, and DFSMSdfp EXPORT and IMPORT. Refer to the documentation on those programs for the most recent information about their use.

# Chapter 8. Estimating disk storage for user data

This chapter provides information on how to estimate the amount of disk storage you need for your data, including:

**Recommendation:** Use DB2 Estimator to calculate space estimates for tables, indexes, and factors discussed in this chapter.

## Factors that affect storage

The amount of disk space you need for your data is not just the number of bytes of data; the true number is some multiple of that. That is,

```
Space required =  M × (bytes of data)
```

The multiplier M depends on your circumstances. It includes factors that are common to all data sets on disk, as well as others that are peculiar to DB2. It can vary significantly, from a low of about 1.25, to 4.0 or more. For a first approximation, set M=2, and skip to “Calculating the space required for a table” on page 84.

For more accuracy, calculate M as the product of the following factors:
- Record overhead
- Free space
- Unusable space
- Data set excess
- Indexes

***Record overhead:*** Allows for eight bytes of record header and control data, plus space wasted for records that do not fit exactly into a DB2 page. For the second consideration, see “Choosing a page size” on page 43. The factor can range from about 1.01 (for a careful space-saving design) to as great as 4.0. A typical value is about 1.10.

***Free space:*** Allows for space intentionally left empty to allow for inserts and updates. You can specify this factor on the CREATE TABLESPACE statement; see “Specifying free space on pages” on page 538 for more information. The factor can range from 1.0 (for no free space) to 200 (99% of each page used left free, and a free page following each used page). With default values, the factor is about 1.05.

***Unusable space:*** Track lengths in excess of the nearest multiple of page lengths. By default, DB2 uses 4 KB pages, which are blocked to fit as many pages as possible on a track. Table 15shows the track size, number of pages per track, and the value of the unusable-space factor for several different device types.

*Table 15. Unusable space factor by device type*

| Device type | 3380 | 3390 | 9340 |
|---|---|---|---|
| Track size | 47476 | 56664 | 46456 |
| Pages per track | 10 | 12 | 10 |
| Factor value | 1.16 | 1.15 | 1.03 |

***Data set excess:*** Allows for unused space within allocated data sets, occurring as unused tracks or part of a track at the end of any data set. The amount of unused space depends upon the volatility of the data, the amount of space management done, and the size of the data set. Generally, large data sets can be managed more closely, and those that do not change in size are easier to manage. The factor can range without limit above 1.02. A typical value is 1.10.

***Indexes:*** Allows for Storage for indexes to data. For data with no indexes, the factor is 1.0. For a single index on a short column, the factor is 1.01. If every column is indexed, the factor can be greater than 2.0. A typical value is 1.20. For further discussion of the factor, see “Calculating the space required for an index” on page 88.

Table 16 shows calculations of the multiplier M for three different database designs:

- The *tight* design is carefully chosen to save space and allows only one index on a single, short field.
- The *loose* design allows a large value for every factor, but still well short of the maximum. Free space adds 30% to the estimate, and indexes add 40%.
- The *medium* design has values between the other two. You might want to use these values in an early stage of database design.

In each design, the device type is assumed to be a 3390. Therefore, the unusable-space factor is 1.15. M is always the product of the five factors.

*Table 16. Calculations for three different database designs*

| Factor | Tight design | Medium design | Loose design |
|---|---|---|---|
| Record overhead × | 1.02 | 1.10 | 1.30 |
| Free space × | 1.00 | 1.05 | 1.30 |
| Unusable space × | 1.15 | 1.15 | 1.15 |
| Data set excess × | 1.02 | 1.10 | 1.30 |
| Indexes = | 1.02 | 1.20 | 1.40 |
| Multiplier M | 1.22 | 1.75 | 3.54 |

In addition to the space for your data, external storage devices are required for:

- Image copies of data sets, which can be on tape
- System libraries, system databases, and the system log
- Temporary work files for utility and sort jobs

A rough estimate of the additional external storage needed is three times the amount calculated above (space for your data) for disk storage.

# Calculating the space required for a table

This section helps you calculate the space required for a table. Space allocation parameters are specified in kilobytes (KB).

## Calculating record lengths and pages

An important consideration in the design of a table is the record size. In DB2, a record is the storage representation of a row. Records are stored within pages that are 4 KB, 8 KB, 16 KB, or 32 KB. Generally, you cannot create a table in which the maximum record size is greater than the page size.

Also consider:
- Normalizing your entities
- Using larger page sizes
- Using LOB data types if a single column in a table is greater than 32 K

In addition to the bytes of actual data in the row (not including LOB data, which is not stored in the base row or included in the total length of the row), each record has:
- A six-byte prefix
- One additional byte for each column that can contain null values
- Two additional bytes for each varying-length column or ROWID column
- Six bytes of descriptive information in the base table for each LOB column

The sum of each column's length is the *record length*, which is the length of data that is physically stored in the table. The *logical record length* can be longer, for example, if the table contains LOBs.

Every data page has:
- A 22-byte header
- A 2-byte directory entry for each record stored in the page

To simplify the calculation of record and page length, consider the directory entry as part of the record. Then, every record has a fixed overhead of 8 bytes, and the space available to store records in a 4 KB page is 4074 bytes. Achieving that maximum in practice is not always simple. For example, if you are using the default values, the LOAD utility leaves approximately 5 percent of a page as free space when loading more than one record per page. Therefore, if two records are to fit in a page, each record cannot be longer than 1934 bytes (approximately 0.95 × 4074 × 0.5). Furthermore, the page size of the table space in which the table is defined limits the record length. If the table space is 4 KB, the record length of each record cannot be greater than 4056 bytes. Because of the 8-byte overhead for each record, the sum of column lengths cannot be greater than 4048 bytes (4056 minus the 8-byte overhead for a record).

DB2 provides three larger page sizes to allow for longer records. You can improve performance by using pages for record lengths that best suit your needs. For details on selecting an appropriate page size, see "Choosing a page size" on page 43.

As shown in Table 17, the maximum record size for each page size depends on the size of the table space and on whether you specified the EDITPROC clause.

*Table 17. Maximum record size (in bytes)*

| EDITPROC | Page Size = 4 KB | Page Size = 8 KB | Page Size = 16 KB | Page Size = 32 KB |
|---|---|---|---|---|
| NO | 4056 | 8138 | 16330 | 32714 |
| YES | 4046 | 8128 | 16320 | 32704 |

Creating a table using CREATE TABLE LIKE in a table space of a larger page size changes the specification of LONG VARCHAR to VARCHAR and LONG VARGRAPHIC to VARGRAPHIC. You can also use CREATE TABLE LIKE to create a table with a smaller page size in a table space if the maximum record size is within the allowable record size of the new table space.

## Saving space with data compression

You can reduce the space required for a table by using data compression if your system meets the requirements. To find out how much space you can save by compressing your data, run the DSN1COMP utility on your data sets. Message DSN1940I of DSN1COMP reports an estimate of the percentage of kilobytes that would be saved by using data compression. See Part 3 of *DB2 Utility Guide and Reference* for more information on the DSN1COMP utility.

The disk saved by data compression is countered by the disk required for a *dictionary*. Every compressed table space or partition requires a dictionary. See “Calculating the space required for a dictionary” on page 87 to figure the disk requirements and the virtual storage requirements for a dictionary.

## Estimating storage for LOBs

Tables with large object data types (LOBs) can store byte strings up to 2 GB. A base table can be defined with one or more LOB columns. The LOB columns are logically part of the base table but are physically stored in an auxiliary table. In place of each LOB column, there is an *indicator column*, which is a column with descriptive information about the LOB. When a base table has LOB columns, then each row of the table has a *row identifier*, which is a varying-length 17-byte field. You must consider the overhead of the indicator column and row identifiers when estimating table size. If the LOB column is NULL or has a value of zero, no space is allocated in the auxiliary table.

When estimating the storage required for LOB table spaces, begin with your estimates from other table spaces, round up to the next page size, and then multiply by 1.1. One page never contains more than one LOB. When a LOB value is deleted, the space occupied by that value remains allocated as long as any application might access that value.

An auxiliary table resides in a LOB table space. There can be only one auxiliary table in a LOB table space. An auxiliary table can store only one LOB column of a base table and there must be one and only one index on this column.

See *DB2 Installation Guide* for information on storage options for LOB values.

## Estimating storage when using the LOAD utility

For a table to be loaded by the LOAD utility, the value can be estimated as follows:

- Let FLOOR be the operation of discarding the decimal portion of a real number.
- Let CEILING be the operation of rounding a real number up to the next highest integer.
- Let *number of records* be the total number of records to be loaded.
- Let *average record size* be the sum of the lengths of the fields in each record, using an average value for varying-length fields, and including the following amounts for overhead:

  8 bytes for the total record
  1 byte for each field that allows nulls
  2 bytes for each varying-length field

  If the average record size is less than 32, use 32. See the CREATE TABLE statement in Chapter 5 of *DB2 SQL Reference* for information on how many bytes are required for different column types.
- Let *percsave* be the percentage of kilobytes saved by compression (as reported by the DSN1COMP utility in message DSN1940I)

- Let *compression ratio* be *percsave*/100

Then calculate as follows:

1. *Usable page size* is the page size minus a number of bytes of overhead (that is, 4 KB− 40 for 4 KB pages, 8 KB− 54 for 8 KB pages, 16 KB−54 for 16 KB pages, or 32 KB−54 for 32 KB pages) multiplied by (100-*p*) / 100, where *p* is the value of PCTFREE. If your average record size is less than 16, then usable page size is 255 (maximum records per page) multiplied by average record size multiplied by (100-*p*) / 100.
2. *Records per page* is MIN(MAXROWS, FLOOR(*usable page size* / *average record size*)), but cannot exceed 255 and cannot exceed the value you specify for MAXROWS.
3. *Pages used* is 2+CEILING(*number of records* / *records per page*).
4. *Total pages* is FLOOR(*pages used* × (1+*fp* ) / *fp* ), where *fp* is the (nonzero) value of FREEPAGE. If FREEPAGE is 0, then *total pages* is equal to *pages used*. (See "Free space" on page 83 for more information about FREEPAGE.) If you are using data compression, you need additional pages to store the dictionary. See "Calculating the space required for a dictionary" to figure how many pages the dictionary requires.
5. Estimated number of kilobytes required for a table:
   - **If you do not use data compression**, the estimated number of kilobytes is *total pages* × page size (4 KB, 8 KB, 16 KB, or 32 KB).
   - **If you use data compression**, the estimated number of kilobytes is (*total pages* × page size (4 KB, 8 KB, 16 KB, or 32 KB) × (1 - *compression ratio*).

For example, consider a table space containing a single table with the following characteristics:
*Number of records* = 100000
*Average record size* = 80 bytes
Page size = 4 KB
PCTFREE = 5 (5% of space is left free on each page)
FREEPAGE = 20 (one page is left free for each 20 pages used)
MAXROWS = 255

If the data is not compressed, you get the following results:
*Usable page size* = 4074 × 0.95 = 3870 bytes
*Records per page* = MIN(MAXROWS, FLOOR(3870 / 80)) = 48
*Pages used* = 2 + CEILING(100000 / 48) = 2085
*Total pages* = FLOOR(2085 ×21 / 20) = 2189
*Estimated number of kilobytes* = 2189 × 4 = 8756

If the data is compressed, multiply the estimated number of kilobytes for an uncompressed table by (1 - *compression ratio*) for the estimated number of kilobytes required for the compressed table.

## Calculating the space required for a dictionary

This section helps you calculate the disk space required by a *dictionary* and the virtual storage required in the DSN1DBM1 address space when a dictionary is read into storage from a buffer pool. A dictionary contains the information used for compressing and decompressing the data in a table space or partition, and it resides in that table space or partition. You can skip this section if you are not going to compress data. Space allocation parameters are specified in pages (either 4 KB, 8 KB, 16 KB, or 32 KB).

## Disk requirements

This section helps you calculate the disk requirements for a dictionary associated with a compressed nonsegmented table space and for a dictionary associated with a compressed segmented table space.

**For a nonsegmented table space**, the dictionary contains 4096 entries in most cases. This means you need to allocate an additional sixteen 4 KB pages, eight 8 KB pages, four 16 KB pages, or two 32 KB pages. Although it is possible that your dictionary can contain fewer entries, allocate enough space to accommodate a dictionary with 4096 entries. For 32 KB pages, 1 segment (minimum of 4 pages) is sufficient to contain the dictionary. Refer to Table 18 to see how many 4 KB pages, 8 KB pages, 16 KB pages, or 32 KB pages to allocate for the dictionary of a compressed nonsegmented table space.

*Table 18. Pages required for the dictionary of a compressed non-segmented table space*

| **Table space page size (KB)** | **Dictionary size (number of entries)** | | | | |
|---|---|---|---|---|---|
| | **512** | **1024** | **2048** | **4096** | **8192** |
| 4 | 2 | 4 | 8 | 16 | 32 |
| 8 | 1 | 2 | 4 | 8 | 16 |
| 16 | 1 | 1 | 2 | 4 | 8 |
| 32 | 1 | 1 | 1 | 2 | 4 |

**For a segmented table space**, the size of the dictionary depends on the size of your segments. Assuming 4096 entries is recommended. Use Table 19 to see how many 4-KB pages to allocate for the dictionary of a compressed segmented table space.

*Table 19. Pages required for the dictionary of a compressed segmented table space*

| **Segment size (4-KB pages)** | **Dictionary size (number of entries)** | | | | |
|---|---|---|---|---|---|
| | **512** | **1024** | **2048** | **4096** | **8192** |
| 4 | 4 | 4 | 8 | 16 | 32 |
| 8 | 8 | 8 | 8 | 16 | 32 |
| 12 | 12 | 12 | 12 | 24 | 36 |
| ≥16 | Segment size | Segment size | Segment size | Segment size | Segment size |

## Virtual storage requirements

You can calculate how much storage is needed in the DSN1DBM1 address space for each dictionary with this formula:

*dictionary size (number of entries)* ×16 bytes

When a dictionary is read into storage from a buffer pool, the *whole* dictionary is read, and it remains there as long as the compressed table space is being accessed.

# Calculating the space required for an index

This section describes the levels of index pages and helps you calculate the storage required for an index.

## Levels of index pages

Indexes can have more than one level of pages. Index pages that point directly to the data in your tables are called *leaf pages*. If the index has more than one leaf page, it must have at least one nonleaf page that contains entries that point to the leaf pages. If the index has more than one nonleaf page, then the nonleaf pages whose entries point directly to leaf pages are said to be on *level 1*; a second level of nonleaf pages must point to level 1, and so on. The highest level contains a single page (which DB2 creates when it first builds your index) called the *root page*. The root page is a 4-KB index page. The index tree points directly to the data in your tables, and gives the key and the RID. Figure 5 shows, in schematic form, a typical index.

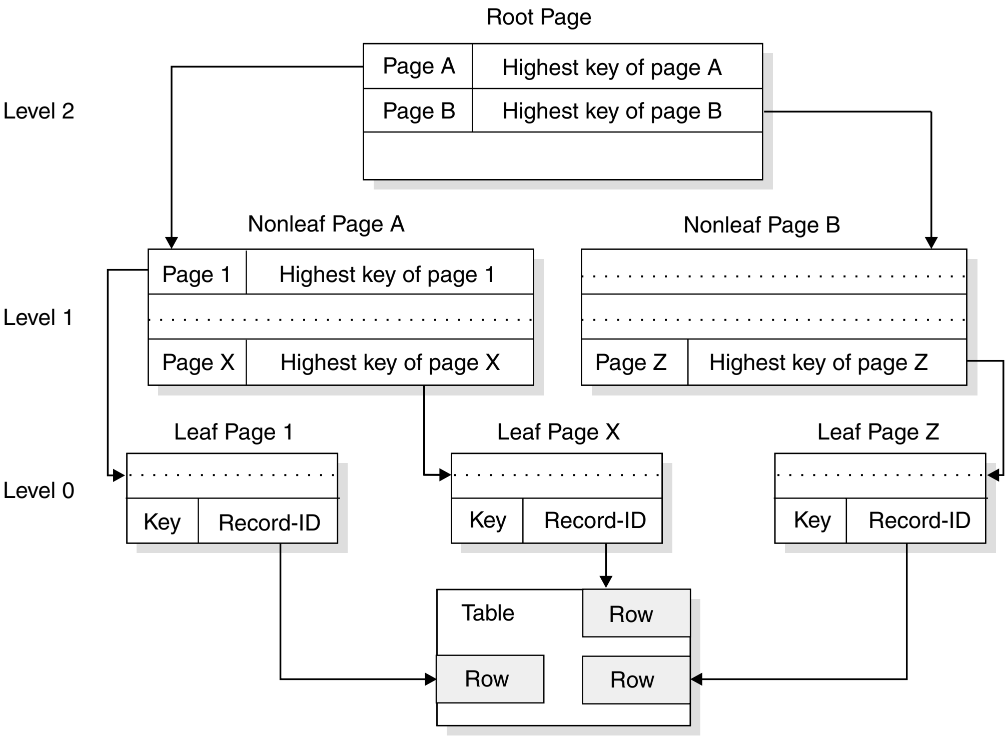


*Figure 5. Sample index structure and pointers (three-level index)*

If you insert data with a constantly increasing key, DB2 adds the new highest key to the top of a new page. Be aware, however, that DB2 treats nulls as the highest value. When the existing high key contains a null value in the first column that differentiates it from the new key that is inserted, the inserted nonnull index entries cannot take advantage of the *highest-value* split.

For example, assume that the existing high key is:

```
SMITH ROBERT J
```

Next you insert:

```
SMITH ROBERT (null)
```

Finally you insert:

```
SMITH ROBERT Z
```

DB2 does not treat this final value as the new high key.

## Calculating the space required for an index

Space allocation parameters are specified in kilobytes. For an index to be loaded by the LOAD utility, you should estimate the future storage requirements of the index.

Estimating the space requirements for DB2 objects is easier if you collect and maintain a statistical history of those objects. The accuracy of your estimates depends on the currentness of the statistical data. To ensure that the statistics history is current, use the MODIFY STATISTICS utility to delete outdated statistical data from the catalog history tables.

The storage required for an index, newly built by the LOAD utility, depends on the number of index pages at all levels. That, in turn, depends on whether the index is unique or not. The numbers of leaf pages (index pages that point directly to the data in your tables) and of nonleaf pages (index pages that contain the page number and the highest key of each page in the next-level index) are calculated separately.

An index key on an auxiliary table used for LOBs is 19 bytes and uses the same formula as other indexes. The RID value stored within the index is 5 bytes, the same as for large table spaces.

These index calculations are intended only to help you estimate the storage required for an index. Because there is no way to predict the exact number of duplicate keys that can occur in an index, the results of these calculations are not absolute. It is possible, for example, that for a nonunique index, more index entries than the calculations indicate might fit on an index page.

The calculations are divided into cases using a unique index and using a nonunique index.

In the following calculations, let:

- *k* = the length of the index key. The length of the index key is the sum of the lengths of all the columns of the key, plus the number of columns that allow nulls.
- *n* = the average number of data records per distinct key value of a nonunique index. For example:

  *a* = number of data records per index
  *b* = number of distinct key values per index
  *s* = the proportion of available space (equal to (100-*f*)/100, where *p* is the value of PCTFREE)
  *n* = a / b
- *f* = the value of PCTFREE.
- *p* = the value of FREEPAGE.
- *r* = record identifier (RID) length. Let *r* = 4 for indexes on non-large table spaces and *r* = 5 for indexes on large spaces and auxiliary tables.
- FLOOR = the operation of discarding the decimal portion of a real number.
- CEILING = the operation of rounding a real number up to the next highest integer.
- MAX = the operation of selecting the highest integer value.

***Calculate pages for a unique index:*** Use the following calculations to estimate the number of leaf and nonleaf pages in a unique index.

Calculate the *total leaf pages*:

1. *Space per key* ≈ *k* + *r* + 3
2. *Usable space per page* ≈FLOOR((100 - *f*) × 4038 / 100)
3. *Entries per page* ≈FLOOR(*usable space per page* / *space per key*)

4. ***Total leaf pages*** ≈CEILING(*number of table rows* / *entries per page*)

Calculate the *total nonleaf pages*:

1. *Space per key* ≈ *k* + 7
2. *Usable space per page* ≈FLOOR (MAX(90, (100- *f*)) × 4046/100)
3. *Entries per page* ≈FLOOR((*usable space per page* / *space per key*)
4. *Minimum child pages* ≈MAX(2, (entries per page + 1))
5. *Level 2 pages* ≈CEILING(*total leaf pages* / *minimum child pages*)
6. *Level 3 pages* ≈CEILING(*level 2 pages* / *minimum child pages*)
7. *Level x pages* ≈CEILING(*previous level pages* / *minimum child pages*)
8. ***Total nonleaf pages*** ≈(*level 2 pages* + *level 3 pages* + ...+ *level x pages* until the number of *level x pages* = 1)

***Calculate pages for a nonunique index:*** Use the following calculations to estimate the number of leaf and nonleaf pages for a nonunique index.

Calculate the *total leaf pages*:

1. *Space per key* ≈ 4 + *k* + (n × *(r+1)*)
2. *Usable space per page* ≈FLOOR((100 - *f*) × 4038 / 100)
3. *Key entries per page* ≈ *n* ×(*usable space per page* / *space per key*)
4. *Remaining space per page* ≈ *usable space per page* - (*key entries per page* / n) × *space per key*
5. *Data records per partial entry* ≈ FLOOR((*remaining space per page* - (*k* + 4)) / 5)
6. *Partial entries per page* ≈(*n* / CEILING(*n* / *data records per partial entry*)) if *data records per partial entry* >= 1, or 0 if *data records per partial entry* < 1
7. *Entries per page* ≈MAX(1, (*key entries per page* + *partial entries per page*))
8. ***Total leaf pages*** ≈CEILING(*number of table rows* / *entries per page*)

Calculate the *total nonleaf pages*:

1. *Space per key* ≈ *k* + *r* + 7
2. *Usable space per page* ≈FLOOR (MAX(90, (100- *f*)) × (4046/100)
3. *Entries per page* ≈FLOOR((*usable space per page* / *space per key*)
4. *Minimum child pages* ≈MAX(2, (entries per page + 1))
5. *Level 2 pages* ≈CEILING(*total leaf pages* / *minimum child pages*)
6. *Level 3 pages* ≈CEILING(*level 2 pages* / *minimum child pages*)
7. *Level x pages* ≈CEILING(*previous level pages* / *minimum child pages*)
8. ***Total nonleaf pages*** ≈(*level 2 pages* + *level 3 pages* + ...+ *level x pages* until *x* = 1)

***Calculate the total space requirement:*** Finally, calculate the number of kilobytes required for an index built by LOAD.

1. *Free pages* ≈ FLOOR(*total leaf pages* / *p*), or 0 if *p* = 0
2. *Space map pages* ≈CEILING((*tree pages* + *free pages*) / 8131)
3. *Tree pages* ≈ MAX(2, (*total leaf pages* + *total nonleaf pages*))
4. *Total index pages* ≈MAX(4, (1 + *tree pages* + *free pages* + *space map pages*))
5. *Total space requirement* ≈4 × (*total index pages* + 2)

In the following example of the entire calculation, assume that an index is defined with these characteristics:

- It is unique.
- The table it indexes has 100000 rows.
- The key is a single column defined as CHAR(10) NOT NULL.
- The value of PCTFREE is 5.
- The value of FREEPAGE is 4.

The calculations are shown in Table 20 on page 92.

*Table 20. The total space requirement for an index*

| Quantity | Calculation | Result |
|---|---|---|
| Length of key | *k* | 10 |
| Average number of duplicate keys | *n* | 1 |
| PCTFREE | *f* | 5 |
| FREEPAGE | *p* | 4 |
| **Calculate total leaf pages** | | |
| Space per key | *k* + 7 | 17 |
| Usable space per page | FLOOR((100 − *f* ) × 4038/100) | 3844 |
| Entries per page | FLOOR(*usable space per page* / *space per key*) | 225 |
| Total leaf pages | CEILING(*number of table rows* / *entries per page*) | 445 |
| **Calculate total nonleaf pages** | | |
| Space per key | *k* + 7 | 17 |
| Usable space per page | FLOOR(MAX(90, (100 − *f* )) × (4046/100) | 3836 |
| Entries per page | FLOOR(*usable space per page* / *space per key*) | 226 |
| Minimum child pages | MAX(2, (*entries per page* + 1)) | 227 |
| Level 2 pages | CEILING(*total leaf pages* / *minimum child pages*) | 2 |
| Level 3 pages | CEILING(*level 2 pages* / *minimum child pages*) | 1 |
| Total nonleaf pages | (*level 2 pages* + *level 3 pages* + ... + *level x pages* until *x* = 1) | 3 |
| **Calculate total space required** | | |
| Free pages | FLOOR(*total leaf pages* / *p*), or 0 if *p* = 0 | 111 |
| Tree pages | MAX(2, (*total leaf pages* + *total nonleaf pages*)) | 448 |
| Space map pages | CEILING((*tree pages* + *free pages*)/8131) | 1 |
| Total index pages | MAX(4, (1 + *tree pages* + *free pages* + *space map pages*)) | 561 |
| TOTAL SPACE REQUIRED, in KB | 4 × (*total index pages* + 2) | 2252 |

# Part 3. Security and auditing

# Chapter 9. Introduction to security and auditing in DB2

The two topics of *security* and *auditing* overlap a great deal, but not completely.

*Security* covers anything to do with the control of access, whether to the DB2 subsystem, its data, or its resources. A security plan sets objectives for a security system, determining who shall have access to what, and in what circumstances. The plan also describes how to meet the objectives, using functions of DB2, functions of other programs, and administrative procedures.

*Auditing* is how you determine whether the security plan is working and who actually has accessed data. It includes other questions also, such as: Have attempts been made to gain unauthorized access? Is the data in the system accurate and consistent? Are system resources used efficiently?

Because the two topics are not the same, this chapter suggests different ways to approach the information that follows. For a brief overview of the range of objects that DB2 protects, look at the left-hand columns of Table 21 on page 105 through Table 30 on page 108.

## Security planning

If you have any sensitive data in your DB2 subsystem, you must plan carefully to allow access to the data only as you desire. The plan sets objectives for the access allowed and describes means of achieving the objectives. Clearly, the nature of the plan depends entirely on the data to be protected, and thus, there is no single way to approach the task. Consider the following suggestions:

### If you are new to DB2

Follow these guidelines to learn about security and auditing:

1. Read carefully the introductory section on “Controlling data access” on page 98.
2. Skim chapters “Chapter 10. Controlling access to DB2 objects” on page 103 through “Chapter 14. Auditing” on page 219. Those chapters describe the tools you use to implement your plan, but they probably contain more detail than you want on a first reading.
3. Read the case study in “Chapter 15. A sample security plan for employee data” on page 233. The sample plan describes decisions of the kind you must make about access to your own data.
4. List your security objectives and the means you will use to achieve them.
5. Reread the chapter parts that describe the functions you expect to use. Be sure you can achieve the objectives you have set, or adjust your plan accordingly.

### If you have used DB2 before

This section contains a summary of the changes in Version 7 for security and auditing.

***Kerberos Security Server:*** You can implement Kerberos authentication through RACF as explained in “Establishing Kerberos authentication through RACF” on page 212.

***Catalog tables for stored procedures:*** Guidelines are given for granting access to catalog tables that programmers need to develop stored procedures in “Controlling access to catalog tables for stored procedures” on page 124.

## Auditing

If you are auditing the activity of a DB2 subsystem, you might have turned directly to this section of your book. If that plunges you into an ocean of unfamiliar terminology, begin by reading “Part 1. Introduction” on page 1, which provides a brief and general view of what DB2 is all about.

We assume you are interested at least in the question of control of access to data. First read “Controlling data access” below, and then “Chapter 10. Controlling access to DB2 objects” on page 103. Read also “Chapter 14. Auditing” on page 219.

## Controlling data access

Access to data includes, but is not limited to, access by a person engaged in an interactive terminal session. For example, access could be from a program running in batch mode, or an IMS or CICS transaction. Hence, so as not to focus your attention too narrowly, we choose the term *process* to represent all access to data.

As Figure 6 on page 99 suggests, there are several routes from a process to DB2 data, with controls on every route.

One of the ways that DB2 controls access to data is through the use of identifiers. Three types of identifiers are: primary authorization IDs, secondary authorization IDs, and SQL IDs.

- Generally it is the primary authorization ID that identifies a process. For example, statistics and performance trace records use a primary authorization ID to identify a process.
- A secondary authorization ID, which is optional, can hold additional privileges available to the process. For example, a secondary authorization ID could be a Resource Access Control Facility (RACF) group ID.
- An SQL ID, which holds the privileges exercised when issuing certain dynamic SQL statements, can be set equal to the primary or any of the secondary IDs. If an authorization ID of a process has SYSADM authority, then the process can set its SQL ID to any authorization ID.

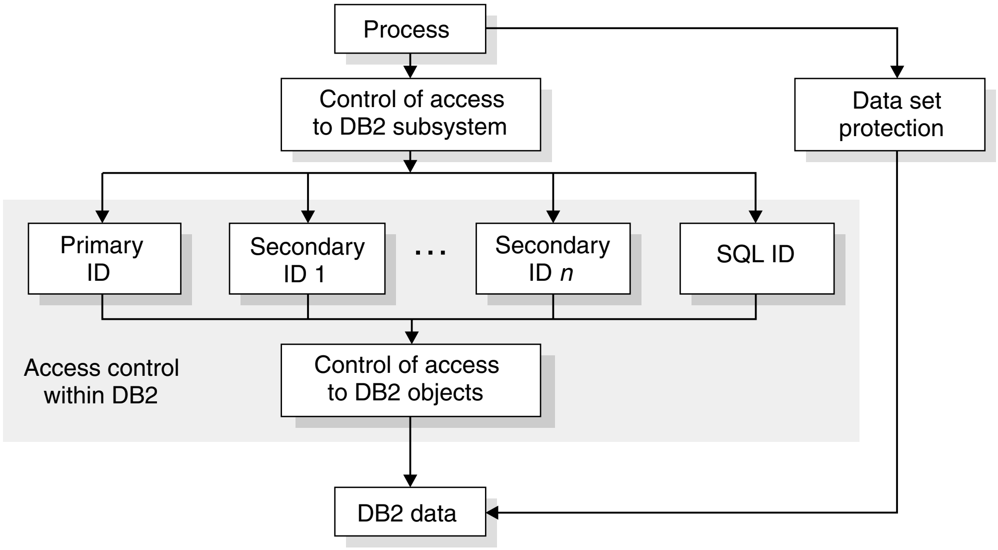

Process
Control of access to DB2 subsystem
Data set protection
Primary ID
Secondary ID 1
. . .
Secondary ID *n*
SQL ID
Access control within DB2
Control of access to DB2 objects
DB2 data

*Figure 6. DB2 data access control*

## Access control within DB2

Within the DB2 subsystem, a process could be represented by a primary authorization identifier (ID), possibly one or more secondary IDs, and an SQL ID. The use of IDs is affected by your security and network systems, and by the choices you make for DB2 connections.

If two different accesses to DB2 are associated with the same set of IDs, DB2 cannot determine whether they involve the same process. You might know that someone else is using your ID, but DB2 does not; nor does DB2 know that you are using someone else's ID. DB2 recognizes only the IDs. Therefore, this book uses phrases like “an ID owns an object” or “taking an action”.

Thus, IDs can hold privileges that allow them to take certain actions or be prohibited from doing so. The list of DB2 privileges provides extremely fine control. For example, you can grant to an ID all the privileges over a table. Or, you could, separately and specifically, grant the privileges to retrieve data from the table, insert rows, delete rows, or update specific columns. By granting or not granting those privileges over views of the table, you can effectively determine exactly what an ID can do to the table, down to the level of specific fields. Specific privileges are also available over databases, plans, packages, and the entire DB2 subsystem.

DB2 also defines sets of related privileges, called *administrative authorities.* By granting an administrative authority to an ID, you grant all the privileges associated with it, in one statement.

Ownership of an object carries with it a set of related privileges over the object. An ID can own an object it creates, or it can create an object to be owned by another ID. There are separate controls for creation and ownership.

The privilege to execute an application plan or a package deserves special attention. Executing a plan or package exercises implicitly all the privileges that the owner needed when binding it. Hence, granting the privilege to execute can provide a finely detailed set of privileges and can eliminate the need to grant other privileges separately. For example, assume an application plan issues the INSERT and SELECT statement on several tables. You need to grant INSERT and SELECT privileges only to the plan owner. Any authorization ID that is later granted the EXECUTE privilege on the plan can perform those same INSERT and SELECT statements through the plan without explicitly being granted the privilege to do so.

Instead of granting privileges to many primary authorization IDs, consider associating each of those primary IDs with the same secondary ID; then, grant the privileges to the secondary ID. A primary ID can be associated with one or more secondary IDs when it gains access to the DB2 subsystem; the association is made within an exit routine. The assignment of privileges to the secondary ID is controlled entirely within DB2.

“Chapter 10. Controlling access to DB2 objects” on page 103 tells how to use the system of privileges within DB2. Alternatively, the entire system of control within DB2 can be disabled, by setting USE PROTECTION to NO when installing or updating DB2. If protection in DB2 is disabled, then any user that gains access can do anything, but no GRANT or REVOKE statements are allowed.

***Using an exit routine to control authorization checking:*** DB2 provides an installation-wide exit point that lets you determine how you want to handle authorization checking. This exit point can give you a single point of control by letting the Security Server of OS/390 Release 4 handle DB2 authorization checks. You can also use this exit point to write your own authorization checking routine. If your installation uses the access control authorization exit, that exit routine might be controlling authorization rules rather then those documented in this publication. For more information about this exit point, see “Access control authorization exit” on page 909.

## Controlling access to a DB2 subsystem

Whether or not a process can gain access to a specific DB2 subsystem can be controlled outside of DB2. A common procedure is to grant access only through RACF or some similar security system. Profiles for access to DB2 from various environments, and DB2 address spaces, are defined as resources to RACF. Each request to access DB2 is associated with an ID. RACF checks that the ID is authorized for DB2 resources and permits, or does not permit, access to DB2.

The RACF system provides several advantages of its own. For example, it can:

- Identify and verify the identifier associated with a process
- Connect those identifiers to RACF group names
- Log and report unauthorized attempts to access protected resources

### Access at a local DB2

A local DB2 user is subject to several checks even before reaching DB2. For example, if you are running DB2 under TSO and using the TSO logon ID as the DB2 primary authorization ID, then that ID was verified with a password when the user logs on.

When the user gains access to DB2, a user-written or IBM-supplied exit routine connected to DB2 can check the authorization ID further, change it, and associate it with secondary IDs. In doing that, it can use the services of an external security system again. “Chapter 12. Controlling access to a DB2 subsystem” on page 169 gives detailed instructions.

### Access from a remote application

A remote user is also subject to several checks before reaching your DB2. You can use RACF or a similar security subsystem.

RACF can:

- Verify an identifier associated with a remote attachment request and check it with a password.
- Generate *PassTickets* on the sending side. PassTickets can be used instead of passwords. A PassTicket lets a user gain access to a host system without sending the RACF password across the network. “Sending RACF PassTickets” on page 197 contains information about RACF PassTickets.

***The communications database:*** DB2’s communications database (CDB) does allow some control of authentication in that you can cause IDs to be translated before sending them to the remote system. See “The communications database for the requester” on page 190 for more information. See “The communications database for the server” on page 178 for information about controls on the server side.

## Data set protection

The data in a DB2 subsystem is contained in data sets. As Figure 6 on page 99 suggests, those data sets might be accessed without going through DB2 at all. If the data is sensitive, you want to control that route.

If you are using RACF or a similar security system to control access to DB2, the simplest means of controlling data set access outside of DB2 is to use RACF for that purpose also. That means defining RACF profiles for data sets and permitting access to them for certain DB2 IDs.

If your data is very sensitive, you may want to consider encrypting it, for protection against unauthorized access to data sets and backup copies outside DB2. You can use DB2 edit procedures or field procedures to encrypt data, and those routines can use the Integrated Cryptographic Service Facility (ICSF) of MVS. For information about this facility, see *ICSF/MVS General Information*.

Data compression is not a substitute for encryption. In some cases, the compression method does not actually shorten the data, and then the data is left uncompressed and readable. If you both encrypt and compress data, compress it first to obtain the maximum compression, and then encrypt the result. When retrieving, take the steps in reverse order: decrypt the data first, and then decompress the result.

# Chapter 10. Controlling access to DB2 objects

The information in this chapter is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1095.

DB2 controls access to its objects by a set of *privileges*. Each privilege allows an action on some object. Figure 7 shows the three primary ways within DB2 to give an ID access to data.[1]

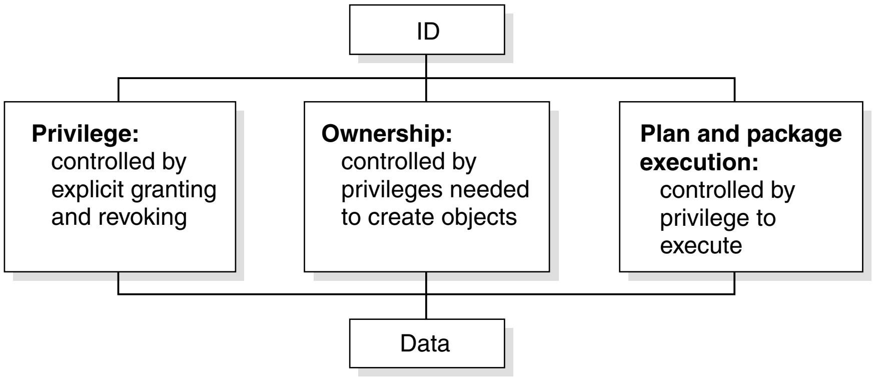

ID
**Privilege:** controlled by explicit granting and revoking
**Ownership:** controlled by privileges needed to create objects
**Plan and package execution:** controlled by privilege to execute
Data

*Figure 7. Access to data within DB2*

The security planner must be aware of every way to allow access to data. To write such a plan, first see:

"Explicit privileges and authorities" on page 104
"Implicit privileges of ownership" on page 114
"Privileges exercised through a plan or a package" on page 117 and "Special considerations for user-defined functions and stored procedures" on page 123.

DB2 has primary authorization IDs, secondary authorization IDs, and SQL IDs. Some privileges can be exercised only by one type of ID, others by more than one. To decide what IDs should hold specific privileges, see "Which IDs can exercise which privileges" on page 129.

After you decide what IDs should hold specific privileges, you have the tools needed to implement a security plan. Before you begin it, see what others have done in "Some role models" on page 139 and "Examples of granting and revoking privileges" on page 140.

Granted privileges and the ownership of objects are recorded in the DB2 catalog. To check the implementation of your security plan, see "Finding catalog information about privileges" on page 152.

The types of objects to which access is controlled are described in "Chapter 2. System planning concepts" on page 7.

[1] Certain authorities are assigned when DB2 is installed, and can be reassigned by changing the subsystem parameter (DSNZPARM); you could consider changing the DSNZPARM value to be a fourth way of granting data access in DB2.

## Explicit privileges and authorities

One way of controlling access within DB2 is by granting, not granting, or revoking explicit privileges and authorities.

A *privilege* allows a specific operation, sometimes on a specific object.

An *explicit privilege* has a name and is held as the result of an SQL GRANT or REVOKE statement.

An *administrative authority* is a set of privileges, often covering a related set of objects. Authorities often include privileges that are not explicit, have no name, and cannot be specifically granted; for example, the ability to terminate any utility job, which is included in the SYSOPR authority.

Privileges and authorities are held by authorization IDs.

## Authorization identifiers

Every process that connects to or signs on to DB2 is represented by a set of one or more DB2 short identifiers called *authorization IDs*. Authorization IDs can be assigned to a process by default procedures or by user-written exit routines. Methods of assigning those IDs are described in detail in “Chapter 12. Controlling access to a DB2 subsystem” on page 169; see especially Table 50 on page 171 and Table 51 on page 172.

As a result of assigning authorization IDs, every process has exactly one ID called the *primary authorization ID*. All other IDs are *secondary authorization IDs*.

Furthermore, one ID (either primary or secondary) is designated as the *current SQL ID*. You can change the value of the SQL ID during your session. If ALPHA is your primary or one of your secondary authorization IDs, you can make it your current SQL ID by issuing the SQL statement:

```
SET CURRENT SQLID = 'ALPHA';
```

If you issue that statement through the distributed data facility, then ALPHA must be one of the IDs associated with your process *at the location where the statement runs*. As explained in “Controlling requests from remote applications” on page 176, your primary ID can be translated before being sent to a remote location, and secondary IDs are associated with your process at the remote location. The current SQL ID, however, is *not* translated.

An ID with SYSADM authority can set the current SQL ID to any string of up to 8 bytes, whether or not it is an authorization ID or associated with the process that is running.

## Explicit privileges

To provide finely detailed control, there are many explicit privileges. The descriptions of the privileges are grouped into categories as follows:

- Tables in Table 21 on page 105
- Plans in Table 22 on page 105
- Packages in Table 23 on page 105
- Collections in Table 24 on page 105
- Databases in Table 25 on page 106
- Systems in Table 26 on page 106
- Usage in Table 27 on page 107
- Schemas in Table 28 on page 107
- Distinct types and Java classes in Table 29 on page 108
- Routines in Table 30 on page 108

*Table 21. Explicit DB2 table privileges*

| Table privileges | Allow these SQL statements for a named table or view |
|---|---|
| ALTER | ALTER TABLE, to change the table definition |
| DELETE | DELETE, to delete rows[2] |
| INDEX | CREATE INDEX, to create an index on the table |
| INSERT | INSERT, to insert rows |
| REFERENCES | ALTER or CREATE TABLE, to add or remove a referential constraint referring to the named table or to a list of columns in the table |
| SELECT | SELECT, to retrieve data from the table |
| TRIGGER | CREATE TRIGGER, to define a trigger on a table |
| UPDATE | UPDATE, to update all columns or a specific list of columns [2] |
| GRANT ALL | SQL statements of all table privileges |

Table 22 shows plan privileges that DB2 allows.

*Table 22. Explicit DB2 plan privileges*

| Plan privileges | Allow these subcommands for a named application plan |
|---|---|
| BIND | BIND, REBIND, and FREE PLAN, to bind or free the plan |
| EXECUTE | RUN, to use the plan when running the application |

Table 23 shows package privileges that DB2 allows.

*Table 23. Explicit DB2 package privileges*

| Package privileges | Allow these operations for a named package |
|---|---|
| BIND | The BIND, REBIND, and FREE PACKAGE subcommands, and the DROP PACKAGE statement, to bind or free the package, and, depending on the installation option BIND NEW PACKAGE, to bind a new version of a package |
| COPY | The COPY option of BIND PACKAGE, to copy a package |
| EXECUTE | Inclusion of the package in the PKLIST option of BIND PLAN |
| GRANT ALL | All package privileges |

Table 24 shows DB2 collection privileges.

*Table 24. Explicit DB2 collection privileges*

| Collection privileges | Allow these operations for a named package collection |
|---|---|
| CREATE IN | Naming the collection in the BIND PACKAGE subcommand |

[2] If you use SQLRULES(STD), or if the CURRENT RULES special register is set to 'STD', you must have the SELECT privilege for searched updates and deletes.

Table 25 shows DB2 database privileges.

*Table 25. Explicit DB2 database privileges*

| Database privileges | Allow these operations on a named database |
|---|---|
| CREATETAB | The CREATE TABLE statement, to create tables in the database |
| CREATETS | The CREATE TABLESPACE statement, to create table spaces in the database |
| DISPLAYDB | The DISPLAY DATABASE command, to display the database status |
| DROP | The DROP and ALTER DATABASE statements, to drop or alter the database |
| IMAGCOPY | The QUIESCE, COPY, and MERGECOPY utilities, to prepare for, make, and merge copies of table spaces in the database; the MODIFY utility, to remove records of copies |
| LOAD | The LOAD utility, to load tables in the database |
| RECOVERDB | The RECOVER, REBUILD INDEX, and REPORT utilities, to recover objects in the database and report their recovery status |
| REORG | The REORG utility, to reorganize objects in the database |
| REPAIR | The REPAIR and DIAGNOSE utilities (except REPAIR DBD and DIAGNOSE WAIT) to generate diagnostic information about, and repair data in, objects in the database |
| STARTDB | The START DATABASE command, to start the database |
| STATS | The RUNSTATS and CHECK utilities, to gather statistics and check indexes and referential constraints for objects in the database |
| STOPDB | The STOP DATABASE command, to stop the database |

Table 26 shows DB2 subsystem privileges.

*Table 26. Explicit DB2 subsystem privileges*

| System privileges | Allow these operations |
|---|---|
| ARCHIVE | The ARCHIVE LOG command, to archive the current active log, the DISPLAY ARCHIVE command, to give information about input archive logs, the SET LOG command, to modify the checkpoint frequency specified during installation, and the SET ARCHIVE command, to control allocation and deallocation of tape units for archive processing. |
| BINDADD | The BIND subcommand with the ADD option, to create new plans and packages |
| BINDAGENT | The BIND, REBIND, and FREE subcommands, and the DROP PACKAGE statement, to bind, rebind, or free a plan or package, or copy a package, on behalf of the grantor. The BINDAGENT privilege is intended for separation of function, not for added security. A bind agent with the EXECUTE privilege might be able to gain all the authority of the grantor of BINDAGENT. |
| BSDS | The RECOVER BSDS command, to recover the bootstrap data set |

*Table 26. Explicit DB2 subsystem privileges (continued)*

| System privileges | Allow these operations |
|---|---|
| CREATEALIAS | The CREATE ALIAS statement, to create an alias for a table or view name |
| CREATEDBA | The CREATE DATABASE statement, to create a database and have DBADM authority over it |
| CREATEDBC | The CREATE DATABASE statement, to create a database and have DBCTRL authority over it |
| CREATEESG | The CREATE STOGROUP statement, to create a storage group |
| CREATETMTAB | The CREATE GLOBAL TEMPORARY TABLE statement, to define a created temporary table |
| DISPLAY | The DISPLAY ARCHIVE, DISPLAY BUFFERPOOL, DISPLAY DATABASE, DISPLAY LOCATION, DISPLAY LOG, DISPLAY THREAD, and DISPLAY TRACE commands, to display system information |
| MONITOR1 | Receive trace data that is not potentially sensitive |
| MONITOR2 | Receive all trace data |
| RECOVER | The RECOVER INDOUBT command, to recover threads |
| STOPALL | The STOP DB2 command, to stop DB2 |
| STOSPACE | The STOSPACE utility, to obtain data about space usage |
| TRACE | The START TRACE, STOP TRACE, and MODIFY TRACE commands, to control tracing |

Table 27 shows DB2 use privileges.

*Table 27. Explicit DB2 use privileges*

| Use privileges | Allow use of these objects |
|---|---|
| USE OF BUFFERPOOL | A buffer pool |
| USE OF STOGROUP | A storage group |
| USE OF TABLESPACE | A table space |

Table 28 shows DB2 schema privileges.

*Table 28. Explicit DB2 schema privileges*

| Schema privileges | Allow use of these operations |
|---|---|
| CREATEIN | Create distinct types, user-defined functions, triggers, and stored procedures in the designated schemas |
| ALTERIN | Alter user-defined functions or stored procedures, or specify a comment for distinct types, user-defined functions, triggers, and stored procedures in the designated schemas |
| DROPIN | Drop distinct types, user-defined functions, triggers, and stored procedures in the designated schemas |

Table 29 shows DB2 distinct type and Java class privileges.

*Table 29. Explicit DB2 distinct type and Java class privileges*

| **Distinct type and Java class privileges** | **Allow use of these objects** |
|---|---|
| USAGE ON DISTINCT TYPE | A distinct type |
| USAGE ON JAR (Java class for a routine) | A Java class |

Table 30 shows DB2 routine privileges.

*Table 30. Explicit DB2 routine privileges*

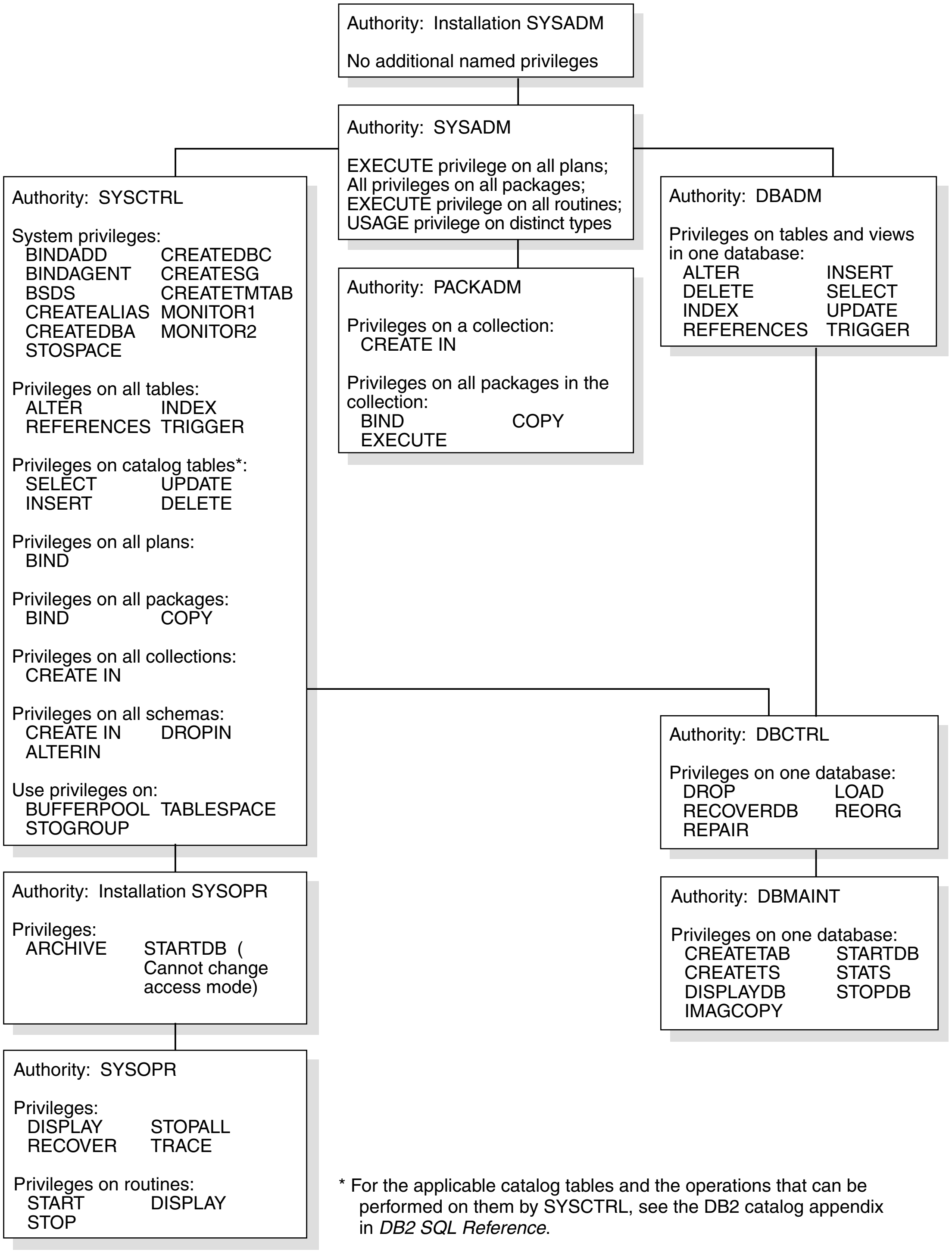
| Routine privileges | Allow use of these objects |
|---|---|
| EXECUTE ON FUNCTION | A user-defined function |
| EXECUTE ON PROCEDURE | A stored procedure |

***Privileges needed for statements, commands, and utility jobs:*** For lists of all privileges and authorities that let you:

- Execute a particular SQL statement, see the description of the statement in Chapter 5 of *DB2 SQL Reference*.
- Issue a particular DB2 command, see the description of the command in Chapter 2 of *DB2 Command Reference*.
- Run a particular type of utility job, see the description of the utility in *DB2 Command Reference*.

## Administrative authorities

Figure 8 on page 109 shows how privileges are grouped into authorities and how the authorities form a branched hierarchy. Table 31 on page 110 supplements the figure and includes capabilities of each authority that are not represented by explicit privileges described in Table 21 on page 105.



* For the applicable catalog tables and the operations that can be performed on them by SYSCTRL, see the DB2 catalog appendix in *DB2 SQL Reference*.

*Figure 8. Individual privileges of administrative authorities. Each authority includes the privileges in its box plus all the privileges of all authorities beneath it. Installation SYSOPR authority is an exception; it can do some things that SYSADM and SYSCTRL cannot.*

Table 31 shows DB2 authorities and the actions that they are allowed.

*Table 31. DB2 authorities*

| Authority | Description |
|---|---|
| SYSOPR | System operator:<br>• Can issue most DB2 commands<br>• *Cannot* issue ARCHIVE LOG, START DATABASE, STOP DATABASE, and RECOVER BSDS<br>• Can terminate any utility job<br>• Can run the DSN1SDMP utility |
| Installation SYSOPR | One or two IDs are assigned this authority when DB2 is installed. They have all the privileges of SYSOPR, plus:<br>• Authority is *not* recorded in the DB2 catalog. The catalog need not be available to check installation SYSOPR authority.<br>• No ID can revoke the authority; it can be removed only by changing the module that contains the subsystem initialization parameters (typically DSNZPARM).<br>Those IDs can also:<br>• Access DB2 when the subsystem is started with ACCESS(MAINT).<br>• Run all allowable utilities on the directory and catalog databases (DSNDB01 and DSNDB06).<br>• Run the REPAIR utility with the DBD statement.<br>• Start and stop the database containing the application registration table (ART) and object registration table (ORT). “Chapter 11. Controlling access through a closed application” on page 157 describes these tables.<br>• Issue dynamic SQL statements that are not controlled by the DB2 governor.<br>• Issue a START DATABASE command to recover objects that have LPL entries or group buffer pool recovery-pending status. These IDs cannot change the access mode. |
| PACKADM | Package administrator, which has all package privileges on all packages in specific collections, or on all collections, plus the CREATE IN privilege on those collections. If held with the GRANT option, PACKADM can grant those privileges to others. If the installation option BIND NEW PACKAGE is BIND, PACKADM also has the privilege to add new packages or new versions of existing packages. |
| DBMAINT | Database maintenance, the holder of which, in a specific database, can create certain objects, run certain utilities, and issue certain commands. If held with the GRANT option, DBMAINT can grant those privileges to others. The holder can use the TERM UTILITY command to terminate all utilities except DIAGNOSE, REPORT, and STOSPACE on the database. |
| DBCTRL | Database control, which includes DBMAINT over a specific database, plus the database privileges to run utilities that can change the data. The user ID with DBCTRL authority can create an alias for another user ID on any table in the database. If held with the GRANT option, DBCTRL can grant those privileges to others. |

*Table 31. DB2 authorities (continued)*

| Authority | Description |
|---|---|
| DBADM | Database administration, which includes DBCTRL over a specific database, plus privileges to access any of its tables through SQL statements. If held with the GRANT option, DBADM can grant those privileges to others.<br>Can also drop and alter any table space, table, or index in the database, issue a COMMENT ON, LABEL ON, or LOCK TABLE statement for any table, and issue a COMMENT ON statement for any index. If the value of field DBADM CREATE VIEW on installation panel DSNTIPP was set to YES during DB2 installation, a user with DBADM authority can:<br>• Create a view for another user ID. The view must be based on at least one table and that table must be in the database where the user ID that issued the CREATE VIEW statement has DBADM authority. See the description of the CREATE VIEW statement in Chapter 5 of *DB2 SQL Reference*.<br>• Create an alias for another user ID on any table in the database.<br>However, a user with DBADM authority on one database can create a view on tables and views in that database and other databases if the authorization ID for which the view is created has all other privileges that are required to create the view. A user with DBADM authority cannot create a view on a view that is owned by another user ID. |
| SYSCTRL | System control, which has nearly complete control of the DB2 subsystem but *cannot* access user data directly, unless granted the privilege to do so. Designed for administering a system containing sensitive data, SYSCTRL can:<br>• Act as installation SYSOPR (when the catalog is available) or DBCTRL over any database<br>• Run any allowable utility on any database<br>• Issue a COMMENT ON, LABEL ON, or LOCK TABLE statement for any table<br>• Create a view for itself or others on any catalog table<br>• Create tables and aliases for itself or others<br>• Bind a new plan or package, naming any ID as the owner<br>Without additional privileges, it *cannot*:<br>• Execute DML statements on user tables or views<br>• Run plans or packages<br>• Set the current SQL ID to a value that is not one of its primary or secondary IDs<br>• Start or stop the database containing the ART and ORT<br>• Act fully as SYSADM or as DBADM over any database<br>• Access DB2 when the subsystem is started with ACCESS(MAINT)<br>SYSCTRL authority is intended for separation of function, not for added security. If any plans have their EXECUTE privilege granted to PUBLIC, an ID with SYSCTRL authority can grant itself SYSADM authority. The only control over such actions is to audit the activity of IDs with high levels of authority. |

*Table 31. DB2 authorities (continued)*

| Authority | Description |
|---|---|
| SYSADM | System administrator, which includes SYSCTRL, plus access to all data. SYSADM can:<br>• Use all the privileges of DBADM over any database<br>• Use EXECUTE and BIND on any plan or package, COPY on any package<br>• Use privileges over views that are owned by others<br>• Set the current SQL ID to any valid value, whether it is currently a primary or secondary authorization ID<br>• Create and drop synonyms and views for others on any table<br>• Use any valid value for OWNER in BIND or REBIND<br>• Drop database DSNDB07<br>• Grant any of the privileges listed above to others<br><br>Holders of SYSADM authority can also drop or alter any DB2 object, except system databases, issue a COMMENT ON or LABEL ON statement for any table or view, and terminate any utility job, but SYSADM cannot specifically grant those privileges. |
| Installation SYSADM | One or two IDs are assigned this authority when DB2 is installed. They have all the privileges of SYSADM, plus:<br>• Authority is *not* recorded in the DB2 catalog. The catalog need not be available to check installation SYSADM authority. (The authority outside the catalog is crucial: If the catalog table space SYSDBAUT is stopped, for example, DB2 cannot check the authority to start it again. Only an installation SYSADM can start it.)<br>• No ID can revoke this authority; it can be removed only by changing the module that contains the subsystem initialization parameters (typically DSNZPARM).<br><br>Those IDs can also:<br>• Run the CATMAINT utility<br>• Access DB2 when the subsystem is started with ACCESS(MAINT)<br>• Start databases DSNDB01 and DSNDB06 when those are stopped or in restricted status<br>• Run the DIAGNOSE utility with the WAIT statement<br>• Start and stop the database containing the ART and ORT |

## Field-level access control by views

Any of the table privileges, except ALTER, REFERENCES, TRIGGER, and INDEX can also be granted on a view. By creating a view and granting privileges on it, you can give an ID access to only a specific combination of data. The capability is sometimes called “field-level access control” or “field-level sensitivity”.

For example, suppose you want a particular ID, say MATH110, to be able to extract certain data from the sample employee table for statistical investigation. To be exact, suppose you want to allow access to data:

- From columns HIREDATE, JOB, EDLEVEL, SEX, SALARY, BONUS, and COMM (but not an employee's name or identification number)
- Only for employees hired after 1975
- Only for employees with an education level of 13 or higher
- Only for employees whose job is *not* MANAGER or PRES

To do that, create and name a view that shows exactly that combination of data:

```
CREATE VIEW SALARIES AS
   SELECT HIREDATE, JOB, EDLEVEL, SEX, SALARY, BONUS, COMM
      FROM DSN8710.EMP
         WHERE HIREDATE > '1975-12-31' AND EDLEVEL >= 13
           AND JOB <> 'MANAGER' AND JOB <> 'PRES';
```

Then grant the SELECT privilege on the view SALARIES to MATH110:

```
GRANT SELECT ON SALARIES TO MATH110;
```

Then MATH110 can execute SELECT statements on the restricted set of data only.

## Authority over the catalog and directory

The DB2 catalog is in database DSNDB06. An ID with SYSCTRL or SYSADM authority can control access to the catalog by granting privileges or authorities on that database or on its tables or views, or by binding plans or packages that access the catalog. Unlike SYSADM, however, SYSCTRL cannot act as DBADM over database DSNDB06.

Authorities that are granted on DSNDB06 also cover database DSNDB01, which contains the DB2 directory. An ID with SYSADM authority can control access to the directory by granting privileges to run utilities (that are listed in Table 32) on DSNDB06, but cannot grant privileges on DSNDB01 directly.

Every authority except SYSOPR carries the privilege to run some utilities on databases DSNDB01 and DSNDB06. Table 32 shows what utilities the other authorities can run on those databases.

*Table 32. Utility privileges on the DB2 catalog and directory*

| **Utilities** | **Authorities** | | |
|---|---|---|---|
| | **Installation SYSOPR, SYSCTRL, SYSADM, Installation SYSADM** | **DBCTRL, DBADM on DSNDB06** | **DBMAINT on DSNDB06** |
| LOAD [1], REPAIR DBD | None (cannot be run on DSNDB01 and DSNDB06) | | |
| CHECK DATA, CHECK LOB, REORG TABLESPACE, STOSPACE | Yes | No | No |
| REBUILD INDEX, RECOVER, REORG INDEX, REPAIR, REPORT | Yes | Yes | No |
| CHECK INDEX, COPY, MERGECOPY, MODIFY, QUIESCE, RUNSTATS | Yes | Yes | Yes |

**Notes:**

1. LOAD can be used to add lines to SYSIBM.SYSSTRINGS.

## Implicit privileges of ownership

You create DB2 objects, except for plans and packages, by issuing SQL CREATE statements in which you name the object. When you create an object, you establish its ownership, and the owner implicitly holds certain privileges over it. (Plans and packages have unique features of their own, described in “Privileges exercised through a plan or a package” on page 117 and “Special considerations for user-defined functions and stored procedures” on page 123.)

## Establishing ownership of objects with unqualified names

If an object name is unqualified, how ownership of the object is established depends on the type of object. Ownership of tables, views, indexes, aliases, and synonyms with unqualified names is established differently than ownership of user-defined functions, stored procedures, distinct types, and triggers with unqualified names. This section describes how ownership is established for each group of objects.

If the name of a table, view, index, alias, or synonym is unqualified, you establish the object's ownership in these ways:

- If you issue the CREATE statement dynamically, perhaps using SPUFI, QMF, or some similar program, the owner of the created object is your current SQL ID. That ID must have the privileges that are needed to create the object.
- If you issue the CREATE statement statically, by running a plan or package that contains it, the ownership of the created object depends on the option used for the bind operation. You can bind the plan or package with either the QUALIFIER option, the OWNER option, or both.
  - If the plan or package is bound with the QUALIFIER option only, the QUALIFIER is the owner of the object. The QUALIFIER option allows the binder to name a qualifier to use for all unqualified names of tables, views, indexes, aliases, or synonyms that appear in the plan or package.
  - If the plan or package is bound with the OWNER option only, the OWNER is the owner of the object.
  - If the plan or package is bound with both the QUALIFIER option and the OWNER option, the QUALIFIER is the owner of the object.
  - If neither option is specified, the binder of the plan or package is implicitly the object owner.

In addition, the plan or package owner must have all required privileges on the objects designated by the qualified names.

If the name of a user-defined function, stored procedure, distinct type, or trigger is unqualified, the implicit qualifier is determined as described in “Qualifying unqualified names” on page 118. However, you establish the ownership of one of these objects in these ways:

- If you issue the CREATE statement dynamically, the owner of the created object is your current SQL ID. That ID must have the privileges that are needed to create the object.
- If you issue the CREATE statement statically, by running a plan or package that contains it, the owner of the object is the plan or package owner. You can use the OWNER bind option to explicitly name the object owner. If you do not use the OWNER bind option, the binder of the package or plan is implicitly the object owner.

The owner of a JAR (Java class for a routine) that is used by a stored procedure or a user-defined function is the current SQL ID of the process that performs the INSTALL_JAR function. For information on installing a JAR, see *DB2 Application Programming Guide and Reference for Java*.

## Establishing ownership of objects with qualified names

If an object name is qualified, how ownership of the object is established depends, again, on the type of object. This section describes how ownership is established for each group of objects.

If you create a table, view, index, or alias with a qualified name, the qualifier becomes the owner of the object, subject to these restrictions for specifying the qualifier:

- If you issue the CREATE statement dynamically, and have no administrative authority, the qualifier must be your primary ID or one of your secondary IDs. However, if your current SQL ID has at least DBCTRL authority, you can use any qualifier for a table or index. If your current SQL ID has at least DBADM authority, it can also use any qualifier for a view.

  If the current SQL ID has at least DBCTRL authority, the qualifier ID does not need any privileges. Otherwise, the SQL ID must have any additional privileges that are needed to create the object; those are CREATETS or USE OF TABLESPACE for a table, and USE OF BUFFERPOOL and USE OF STOGROUP for an index. If the current SQL ID does not have at least DBCTRL authority, all the necessary privileges must be held by the qualifier ID.
- If you issue the CREATE statement statically, and the owner of the plan or package that contains the statement has no administrative authority, the qualifier can be only the owner. However, if the owner has at least DBCTRL authority, the plan or package can use any qualifier for a table or an index. If the owner of the plan or package has at least DBADM authority, it can also use any qualifier for a view.

If you create a distinct type, user-defined function, stored procedure, or trigger with a qualified name, the qualifier is the *schema name*. It identifies the schema to which the object belongs. You can think of all objects that are qualified by the same schema name as a group of related objects. Unlike other objects, however, the qualifier does not identify the owner of the object. You establish ownership of a distinct type, user-defined function, stored procedure, or trigger in these ways:

- If you issue the CREATE statement dynamically, the owner of the created object is your current SQL ID. That ID must have the privileges that are needed to create the object.
- If you issue the CREATE statement statically, by running a plan or package that contains it, the owner of the object is the plan or package owner. You can use the OWNER bind option to explicitly name the object owner. If you do not use the OWNER bind option, the binder of the package or plan is the implicit object owner.

The owner of a JAR (Java class for a routine) that is used by a stored procedure or a user-defined function is the current SQL ID of the process that performs the INSTALL_JAR function. For information on installing a JAR, see *DB2 Application Programming Guide and Reference for Java*

## Privileges by type of object

The following table lists implicit privileges of ownership for each type of object.

*Table 33. Implicit privileges of ownership by object type*

| Object type | Implicit privileges of ownership |
|---|---|
| Storage group | To alter or drop the group and to name it in the USING clause of a CREATE INDEX or CREATE TABLESPACE statement |
| Database | DBCTRL or DBADM authority over the database, depending on the privilege (CREATEDBC or CREATEDBA) that is used to create it. DBCTRL authority does *not* include the privilege to access data in tables in the database. |
| Table space | To alter or drop the table space and to name it in the IN clause of a CREATE TABLE statement |
| Table | • To alter or drop the table or any indexes on it<br>• To lock the table, comment on it, or label it<br>• To create an index or view for the table<br>• To select or update any row or column<br>• To insert or delete any row<br>• To use the LOAD utility for the table<br>• To define referential constraints on any table or set of columns<br>• To create a trigger on the table |
| Index | To alter, comment on, or drop the index |
| View | • To drop, comment on, or label the view, or to select any row or column<br>• To update any row or column, insert or delete any row (if the view is not read-only) |
| Synonym | To use or drop the synonym |
| Package | To bind, rebind, free, copy, execute, or drop the package |
| Plan | To bind, rebind, free, or execute the plan |
| Alias | To drop the alias |
| Distinct type | To use or drop a distinct type |
| User-defined functions | To execute, alter, drop, start, stop, or display a user-defined function |
| Stored procedure | To execute, alter, drop, start, stop, or display a stored procedure |
| JAR (Java class for a routine) | To replace, use, or drop the JAR |

## Granting implicit privileges

Some implicit privileges of ownership correspond to privileges that can be granted by a GRANT statement, and some do not. For those that do correspond, the owner of the object can grant the privilege to another user. For example, the owner of a table can grant the SELECT privilege on the table to any other user.

## Changing ownership

The privileges that are implicit in ownership cannot be revoked. Except for a plan or package, as long as an object exists, its owner cannot be changed. All that can be done is to drop the object, which usually deletes all privileges on it, and then re-create it with a new owner.[3]

[3] Dropping a package does not delete all privileges on it if another version of the package still remains in the catalog.

In practice, however, sharing the privileges of ownership is sometimes appropriate. To do this, make the owning ID a secondary ID to which several primary authorization IDs are connected. You can change the list of primary IDs connected to the secondary ID without dropping and re-creating the object.

## Privileges exercised through a plan or a package

This section describes the privileges that are required for executing plans and packages. User-defined function and stored procedure packages, also known as *routine* packages, have additional, unique requirements that are described in “Special considerations for user-defined functions and stored procedures” on page 123.

An application plan or a package can take many actions on many tables, all of them requiring one or more privileges. The owner of the plan or package must hold every required privilege. Another ID can execute the plan or package if it has only the EXECUTE privilege. In that way, another ID can exercise all the privileges that are used in validating the plan or package, but only within the restrictions that are imposed by the SQL statements in the original program.

For example, the program might contain:

```
EXEC SQL
  SELECT * INTO :EMPREC FROM DSN8710.EMP
    WHERE EMPNO='000010';
```

The example puts the data for employee number 000010 into the host structure EMPREC. The data comes from table DSN8710.EMP. However, the ID that has EXECUTE privilege for this plan can only access rows in the DSN8710.EMP table that have EMPNO = '000010'.

The executing ID can use some of the owner's privileges, within limits. If the privileges are revoked from the owner, the plan or the package is invalidated. It must be rebound, and the new owner must have the required privileges.

## Establishing ownership of a plan or a package

The BIND and REBIND subcommands create or change an application plan or a package. On either subcommand, use the OWNER option to name the owner of the resulting plan or package. Keep these points in mind when naming an owner:

- Any user can name the primary or any secondary ID.
- An ID with the BINDAGENT privilege can name the grantor of that privilege.
- An ID with SYSCTRL or SYSADM authority can name any authorization ID on a BIND command, but not on a REBIND command.

If you omit the OWNER option:

- On BIND, your primary ID becomes the owner.
- On REBIND, the previous owner retains ownership.

Some systems that can bind a package at a DB2 system do not support the OWNER option. When the option is not supported, the primary authorization ID is always the owner of the package, and a secondary ID cannot be named as the owner.

## Qualifying unqualified names

A plan or package can contain SQL statements that use unqualified table and view names. For static SQL, the default qualifier for those names is the owner of the plan or package. However, you can use the QUALIFIER option of the BIND command to specify a different qualifier.

For plans or packages that contain static SQL, using the BINDAGENT privilege and the OWNER and QUALIFIER options gives you considerable flexibility in performing bind operations. For example, if ALPHA has the BINDAGENT privilege from BETA, and BETA has privileges over tables that are owned by GAMMA, ALPHA can bind a plan using OWNER (BETA) and QUALIFIER (GAMMA). ALPHA, as merely a binding agent, does not need to have privileges over the tables and does not have the privilege to execute the plan.

For plans or packages that contain dynamic SQL, DYNAMICRULES behavior determines how DB2 qualifies unqualified object names. See “Authorization for dynamic SQL statements” on page 132 for more information.

For unqualified distinct types, user-defined functions, stored procedures, and trigger names in dynamic SQL statements, DB2 finds the schema name to use as the qualifier by searching schema names in the CURRENT PATH special register. For static statements, the PATH bind option determines the path that DB2 searches to resolve unqualified distinct types, user-defined functions, stored procedures, and trigger names.

However, an exception exists for ALTER, CREATE, DROP, COMMENT ON, GRANT, and REVOKE statements. For static SQL, specify the qualifier for these statements in the QUALIFIER bind option. For dynamic SQL, the qualifier for these statements is the authorization ID of the CURRENT SQLID special register. See Chapter 2 of *DB2 SQL Reference* for more information about unqualified names.

## Checking authorization to execute

The plan or package owner must have authorization to execute all static SQL statements that are embedded in the plan or package. However, you do not need to have the authorizations in place when the plan or package is bound; in fact, the SQL objects that are referred to do not need to exist at that time.

A bind operation always checks whether a local object exists and whether the owner has the required privileges on it. Any failure results in a message. To choose whether the failure prevents the bind operation from completing, use the VALIDATE option of BIND PLAN and BIND PACKAGE, and also the SQLERROR option of BIND PACKAGE. See Part 5 of *DB2 Application Programming and SQL Guide* for instructions. If you let the operation complete, the checks are made again at run time. The corresponding checks for remote objects are always made at run time.

Authorization to execute dynamic SQL statements is also checked at run time. Table 36 on page 129 shows which IDs can supply the authorizations that are required for different types of statements.

Applications that use the Recoverable Resource Manager Services attachment facility (RRSAF) to connect to DB2 do not require a plan. If the requesting application is an RRSAF application, DB2 follows the rules described in “Checking authorization to execute an RRSAF application without a plan” on page 120 to check authorizations.

## Checking authorization at a second DB2 server

Authorization for execution at a second DB2 server (also known as a “double-hop” situation) is a special case of DB2 private protocol access when bind option DBPROTOCOL (PRIVATE) is in effect. See Figure 9.

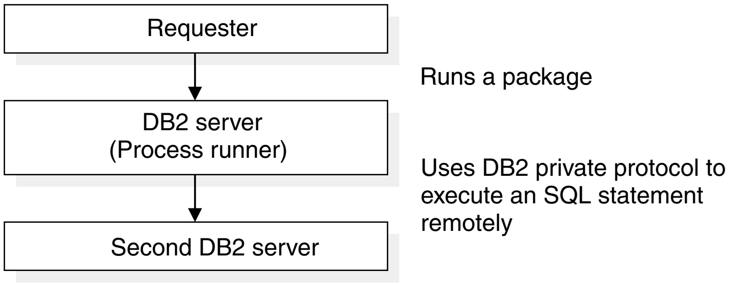

Requester
Runs a package
DB2 server (Process runner)
Uses DB2 private protocol to execute an SQL statement remotely
Second DB2 server

*Figure 9. Execution at a second DB2 server*

In the figure, a remote requester, either a DB2 for OS/390 and z/OS or some other requesting system, runs a package at the DB2 server. A statement in the package uses an alias or a three-part name to request services from a second DB2 for OS/390 and z/OS server. The ID that is checked for the privileges that are needed to run at the second server can be:

- The owner of the plan that is running at the requester (if the requester is DB2 for MVS/ESA or DB2 for OS/390 and z/OS)
- The owner of the package that is running at the DB2 server
- The authorization ID of the process that runs the package at the first DB2 server (the “process runner”)

In addition, if a remote alias is used in the SQL, the alias must be defined at the requester site. The ID that is used depends on these four factors:

- Whether the requester is DB2 for OS/390 and z/OS or DB2 for MVS/ESA, or a different system.
- The value of the bind option DYNAMICRULES. See “Authorization for dynamic SQL statements” on page 132 for detailed information about the DYNAMICRULES options.
- Whether the parameter HOPAUTH at the DB2 server site was set to BOTH or RUNNER when the installation job DSNTIJUZ was run. The default value is BOTH.
- Whether the statement that is executed at the second server is static or dynamic SQL.

***Hop situation with non-DB2 for OS/390 and z/OS or DB2 for MVS/ESA server:*** Using DBPROTOCOL(DRDA), a three-part name statement can hop to a server other than DB2 for OS/390 and z/OS or DB2 for MVS/ESA. In this hop situation, only package authorization information is passed to the second server.

A hop is not allowed on a connection that matches the LUWID of another existing DRDA thread. For example, in a hop situation from site A to site B to site C to site A, a hop is not allowed to site A again.

Table 34 on page 120 shows how these factors determine the ID that must hold the required privileges when bind option DBPROTOCOL (PRIVATE) is in effect.

*Table 34. The authorization ID that must hold required privileges for the double-hop situation*

| Requester | DYNAMICRULES | HOPAUTH | Statement | Authorization ID |
|---|---|---|---|---|
| DB2 for MVS/ESA or DB2 for OS/390 and z/OS | Run behavior (default)* | n/a | Static | Plan owner |
| | | | Dynamic | Process runner |
| | Bind behavior* | n/a | Either | Plan owner |
| Different system or RRSAF application without a plan | Run behavior (default)* | YES (default) | Static | Package owner |
| | | | Dynamic | Process runner |
| | | NO | Either | Process runner |
| | Bind behavior* | n/a | Either | Package owner |

**Note:** *If DYNAMICRULES define behavior is in effect, DB2 converts to DYNAMICRULES bind behavior. If DYNAMICRULES invoke behavior is in effect, DB2 converts to DYNAMICRULES run behavior.

## Checking authorization to execute an RRSAF application without a plan

RRSAF provides the capability for an application to connect to DB2 and run without a DB2 plan. If an RRSAF application does not have a plan, the following authorization rules are true:

- For the following types of packages, the primary or secondary authorization ID of the process is used for checking authorization to execute the package:
  - A local package
  - A remote package that is on a DB2 for OS/390 and z/OS or DB2 for MVS/ESA system and is accessed using DRDA
- At a DB2 for OS/390 and z/OS or DB2 for MVS/ESA server, the authorization to execute the DESCRIBE TABLE statement includes checking the primary and secondary authorization IDs.
- For a double hop situation, the authorization ID that must hold the required privileges to execute SQL statements at the second server is determined as if the requester is not a DB2 for OS/390 and z/OS or DB2 for MVS/ESA system. Table 34 lists the specific privileges.

## Caching authorization IDs for best performance

You can specify that DB2 cache authorization IDs for plans, packages, or routines (user-defined functions and stored procedures). Caching IDs can greatly improve performance, especially when user IDs are reused frequently. One cache exists for each plan, one global cache exists for packages, and a global cache exists for routines. The global cache for packages and routines are allocated at DB2 startup. For a data sharing group, each member does its own authorization caching.

***Caching IDs for plans:*** Authorization checking is fastest when the EXECUTE privilege is granted to PUBLIC and, after that, when the plan is reused by an ID that already appears in the cache.

You set the size of the plan authorization cache in the BIND PLAN subcommand. For suggestions on setting this cache size, see Part 5 of *DB2 Application Programming and SQL Guide*. The default cache size is specified by an installation option, with an initial default setting of 1024 bytes.

***Caching IDs for packages:*** This performance enhancement provides a run-time benefit for:

- Stored procedures.

- Remotely bound packages.
- Local packages in a package list in which the plan owner does not have execute authority on the package at bind time, but does at run time.
- Local packages that are not explicitly listed in a package list, but are implicitly listed by *collection-id.**, *.*, or **.package-id*.

Set the size of the package authorization cache using the PACKAGE AUTH CACHE field on installation panel DSNTIPP. The default value, 32 KB, is enough storage to support about 370 *collection-id.package-id* entries or *collection-id.** entries.

You can cache more package authorization information by granting package execute authority to *collection.**, by granting package execute authority to PUBLIC for some packages or collections, or by increasing the size of the cache.

Field QTPACAUT in the package accounting trace indicates how often DB2 was successful at reading package authorization information from the cache.

***Caching IDs for routines:*** The routine authorization cache stores authorization IDs with the EXECUTE privilege on a specific routine. A routine is identified as schema.routine-name.type, where the routine name is the specific function name for user-defined functions, the procedure name for stored procedures, or '*' for all routines in the schema.

Set the size of the routine authorization cache using the ROUTINE AUTH CACHE field on installation panel DSNTIPP. The initial default setting of 32 KB is enough storage to support about 370 *schema.routine.type* or *schema.*.type* entries.

You can cache more routine authorization information by granting EXECUTE on *schema.**, by granting routine execute authority to PUBLIC for some or all routines in the schema, or by increasing the size of the cache.

## Controls in the program

Because an ID executes a package or an application plan by running a program, implementing control measures in the program can be useful. For example, consider the SQL statement on page 117, which permits access to the row of the employee table `WHERE EMPNO='000010'`. If you replace the value 10 with a host variable, the program could supply the value of the variable and permit access to various employee numbers. Routines in the program could limit that access to certain IDs, or to certain times of the day, on certain days of the week, or in other special circumstances.

Stored procedures provide an alternative to controls in the program. By encapsulating several SQL statements into a single message to the DB2 server, sensitive portions of the application program can be protected. Also, stored procedures can include access to non-DB2 resources, as well as DB2.

### A recommendation against use of controls in the program

Do not use programs to extend security. Whenever possible, use other techniques, such as stored procedures or views, as a security mechanism. Program controls are separate from other access controls, can be difficult to implement properly, are difficult to audit, and relatively simple to bypass. Almost any debugging facility can be used to bypass security checks. Other programs might use the plan without doing the needed checking. Errors in the program checks might allow unauthorized access.

Because the routines that check security might be quite separate from the SQL statement, the security check could be entirely disabled without requiring a bind operation for a new plan.

Also, a BIND REPLACE operation for an existing plan does not necessarily revoke the existing EXECUTE privileges on the plan. (To revoke those privileges is the default, but the plan owner has the option to retain them. For packages, the EXECUTE privileges are always retained.)

For those reasons, if the program accesses any sensitive data, the EXECUTE privileges on the plan and on packages are also sensitive. They should be granted only to a carefully planned list of IDs.

### Restricting a plan or a package to particular systems

If you do use controls in the program, limit the use of a plan or package to the particular systems for which it was designed. DB2 does not ensure that only specific programs are used with a plan, but program-to-plan control can be enforced in IMS and CICS. DB2 does provide a consistency check to avoid accidental mismatches between program and plan, but that is not a security check.

***The ENABLE and DISABLE options:*** The ENABLE and DISABLE options on the BIND and REBIND subcommands for plans and packages can limit their use. For example, ENABLE IMS allows running the plan or package from any IMS connection and, unless other systems are named also, prevents running it from any other type of connection. DISABLE BATCH prevents running the plan or package through a batch job but allows running it from all other types of connection. You can exercise even finer control, enabling or disabling particular IMS connection names, CICS application IDs, requesting locations, and so on. For details, see the syntax of the BIND and REBIND subcommands in *DB2 Command Reference*.

## Privileges required for remote packages

Generally, the privileges that are required for a remote bind (BIND PACKAGE *location.collection*) must be granted at the server location. That is, the ID that owns the package must have all the privileges that are required to run the package at the server, and BINDADD[4] and CREATE IN privileges at the server. The exceptions are:

- For a BIND COPY operation, the owner must have the COPY privilege at the local DB2, where the package being copied resides.
- If the creator of the package is not the owner, the creator must have SYSCTRL authority or higher, or must have been granted the BINDAGENT privilege by the owner. That authority or privilege is granted at the local DB2.

Binding a plan with a package list (BIND PLAN PKLIST) is done at the local DB2, and bind privileges must be held there. Authorization to execute a package at a remote location is checked at execution time, as follows:

- For DB2 private protocol, the owner of the plan at the requesting DB2 must have EXECUTE privilege for the package at the DB2 server.
- For DRDA, if the server is a DB2 for OS/390 and z/OS subsystem, the authorization ID of the process (primary ID or any secondary ID) must have EXECUTE privilege for the package at the DB2 server.
- If the server is not DB2 for OS/390 and z/OS, the primary authorization ID must have whatever privileges are needed. Check that product's documentation.

[4] Or BIND, depending on the installation option BIND NEW PACKAGE.

## Special considerations for user-defined functions and stored procedures

A number of steps are involved in implementing, defining, and invoking user-defined functions and stored procedures, which are also called *routines*. This section explains those steps and the authorizations they require. They are summarized in Table 35.

*Table 35. Common tasks and required privileges for routines*

| Role | Tasks | Required privileges |
|---|---|---|
| Implementor | If SQL is in the routine: codes, precompiles, compiles, and link-edits the program to use as the routine. Binds the program as the routine package.<br>If no SQL is in the routine: codes, compiles, and link-edits the program. | If binding a package, BINDADD system privilege and CREATE IN on the collection. |
| Definer | Issues a CREATE FUNCTION statement to define a user-defined function or CREATE PROCEDURE statement to define a stored procedure. | CREATEIN privilege on the schema. EXECUTE authority on the routine package when invoked. |
| Invoker | Invokes a routine from an SQL application. | EXECUTE authority on the routine. |

The routine *implementor* typically codes the routine in a program, precompiles the program, and binds the DBRM, if the program contains SQL statements. In general, the authorization ID that binds the DBRM into a package is the package owner. The implementor is the routine package owner. As package owner, the implementor has EXECUTE authority (implicitly) on the package and has the authority to grant EXECUTE privileges to other users to execute the code within the package.

The implementor grants EXECUTE authority on the routine package to the definer. EXECUTE authority is only necessary if the package contains SQL. For user-defined functions, the definer requires EXECUTE authority on the package. For stored procedures, EXECUTE authority on the package is not limited to the definer.

The *definer* is the routine owner. The definer issues a CREATE FUNCTION statement to define a user-defined function or a CREATE PROCEDURE statement to define a stored procedure. If the SQL statement is:

- Embedded in an application program, the definer is the authorization ID of the owner of the plan or package.
- Dynamically prepared, the definer is the SQL authorization ID that is contained in the CURRENT SQLID special register.

The definer grants EXECUTE authority on the routine to the invoker, that is, any user ID that needs to invoke the routine.

The *invoker* invokes the routine from an SQL statement in the invoking plan or package. The invoker:

- For a static statement, is the authorization ID of the plan or package owner.
- For a dynamic statement, depends on DYNAMICRULES behavior. See “Authorization for dynamic SQL statements” on page 132 for a description of the options.

See Chapter 5 of *DB2 SQL Reference* for more information about the CREATE FUNCTION and CREATE PROCEDURE statements.

## Additional authorization for stored procedures

Prior to Version 7, stored procedures were defined to DB2 by inserting rows into catalog table SYSIBM.SYSPROCEDURES. Starting in Version 7, a stored procedure is defined using the CREATE PROCEDURE statement.

The CALL statement invokes a stored procedure. The privileges that are required to execute a stored procedure invoked by the CALL statement are described in Chapter 5 of *DB2 SQL Reference*.

This section also describes additional privileges that are required on each package that the stored procedure uses during its execution. The database server determines the privileges that are required and the authorization ID that must have the privileges.

## Controlling access to catalog tables for stored procedures

The catalog tables SYSROUTINES_SRC and SYSROUTINES_OPTS contain source code and build options for generated routines that are created by coding tools like the DB2 Stored Procedure Builder. Because a variety of users can use these coding tools, you need to control access to these catalog tables by performing the following steps:

- Determine criteria for limiting each application programmer's access to a subset of the SYSROUTINES_SRC and SYSROUTINES_OPTS rows.
- Create a view for each programmer by using these criteria.
- Grant the SELECT, INSERT, UPDATE, and DELETE privileges on each view to the appropriate programmer.

For example, programmer A1 is working on a set of stored procedures for project B1. You decide that programmer A1 must use schema names for the stored procedures that begin with the characters A1B1. Then you can create views that limit A1's SYSROUTINES_SRC and SYSROUTINES_OPTS accesses to rows where the SCHEMA value begins with A1B1. The following CREATE statement creates a view on SYSROUTINES_SRC:

```
CREATE VIEW A1.B1GRSRC AS
  SELECT SCHEMA, ROUTINENAME, VERSION,
    SEQNO, IBMREQD, CREATESTMT
    FROM SYSIBM.SYSROUTINE_SRC
    WHERE SCHEMA LIKE 'A1B1%'
WITH CHECK OPTION;
```

The following CREATE statement creates a view on SYSROUTINES_OPTS:

```
CREATE VIEW A1.B1GROPTS AS
  SELECT SCHEMA, ROUTINENAME, VERSION,
  BUILDSCHEMA, BUILDNAME, BUILDOWNER, IBMREQD,
  PRECOMPILE_OPTS, COMPILE_OPTS, PRELINK_OPTS,
  LINK_OPTS, BIND_OPTS, SOURCEDSN
  FROM SYSIBM.SYSROUTINE_OPTS
  WHERE SCHEMA LIKE 'A1B1%'
WITH CHECK OPTION;
```

Finally, use the following statement to let A1 view or update the appropriate SYSROUTINE_SRC and SYSROUTINE_OPTS rows:

```
GRANT SELECT, INSERT, DELETE, UPDATE
  ON (A1.B1GRSRC,A1.B1GROPTS)
  TO A1;
```

After a set of generated routines goes into production, you can decide to regain control over the routine definitions in SYSROUTINES_SRC and SYSROUTINES_OPTS by revoking the INSERT, DELETE, and UPDATE privileges on the appropriate views. It is convenient for programmers to keep the SELECT privilege on their views so that they can use the old rows for reference when they define new generated routines.

## Example of routine roles and authorizations

This example describes how to get a routine up and running, and how to use and assign the required privileges and authorizations. In the example, the routine is an external user-defined function.

### How to code the user-defined function program (implementor role)

1. The implementor codes a user-defined function program that implements the user-defined function. Assume the implementor codes the following external user-defined function, C_SALARY, written in C:

```
/**********************************************************************
* This routine accepts an employee serial number and a percent raise. *
* If the employee is a manager, the raise is not applied.  Otherwise, *
* the new salary is computed, truncated if it exceeds the employee's  *
* manager's salary, and then applied to the database.                *
**********************************************************************/
 void C_SALARY                         /* main routine               */
 ( char        *employeeSerial         /* in: employee serial no.    */
   decimal     *percentRaise           /* in: percentage raise       */
   decimal     *newSalary,             /* out: employee's new salary */
   short int   *niEmployeeSerial       /* in: indic var, empl ser    */
   short int   *niPercentRaise         /* in: indic var, % raise     */
   short int   *niNewSalary,           /* out: indic var, new salary */
   char        *sqlstate,              /* out: SQLSTATE              */
   char        *fnName,                /* in: family name of function*/
   char        *specificName,          /* in: specific name of func  */
   char        *message                /* out: diagnostic message    */
 )
 {
   EXEC SQL BEGIN DECLARE SECTION;
   char        hvEMPNO-7-;             /* host var for empl serial   */
   decimal     hvSALARY;               /* host var for empl salary   */
   char        hvWORKDEPT-3-;          /* host var for empl dept no. */
   decimal     hvManagerSalary;        /* host var, emp's mgr's salry*/
   EXEC SQL END DECLARE SECTION;

   sqlstate = 0;
   memset( message,0,70 );
   /*******************************************************************
   * Copy the employee's serial into a host variable                  *
   *******************************************************************/
   strcpy( hvEMPNO,employeeSerial );
   /*******************************************************************
   * Get the employee's work department and current salary            *
   *******************************************************************/
   EXEC SQL SELECT  WORKDEPT, SALARY
              INTO :hvWORKDEPT, :hvSALARY
              FROM  EMP
             WHERE  EMPNO = :hvEMPNO;
   /*******************************************************************
   * See if the employee is a manager                                 *
   *******************************************************************/
   EXEC SQL SELECT  DEPTNO
              INTO :hvWORKDEPT
              FROM  DEPT
             WHERE  MGRNO = :hvEMPNO;
   /*******************************************************************
   * If the employee is a manager, do not apply the raise             *
   *******************************************************************/
   if( SQLCODE == 0 )
     {
       newSalary = hvSALARY;
     }
   /*******************************************************************
   * Otherwise, compute and apply the raise such that it does not     *
   * exceed the employee's manager's salary                           *
   *******************************************************************/
   else
     {
       /***************************************************************
       * Get the employee's manager's salary                          *
       ***************************************************************/
       EXEC SQL SELECT  SALARY
                  INTO :hvManagerSalary
                  FROM  EMP
                 WHERE  EMPNO = (SELECT MGRNO
                                  FROM DSN8610.DEPT
                                 WHERE DEPTNO = :hvWORKDEPT);
       /***************************************************************
       * Compute proposed raise for the employee                      *
       ***************************************************************/
       newSalary = hvSALARY * (1 + percentRaise/100);
       /***************************************************************
       * Don't let the proposed raise exceed the manager's salary     *
       ***************************************************************/
       if( newSalary > hvManagerSalary
         newSalary = hvManagerSalary;
       /***************************************************************
       * Apply the raise                                              *
       ***************************************************************/
       hvSALARY = newSalary;
       EXEC SQL UPDATE  EMP
                   SET  SALARY = :hvSALARY
                 WHERE  EMPNO  = :hvEMPNO;
     }

   return;
} /* end C_SALARY */
```

The implementor requires the UPDATE privilege on table EMP. Users with the EXECUTE privilege on function C_SALARY do not need the UPDATE privilege on the table.

2. Because this function program contains SQL, the implementor performs the following steps:
   - Precompiles the user-defined function program
   - Link-edits the user-defined function program with DSNRLI (RRS attachment facility) and names the user-defined function program’s load module C_SALARY
   - Binds the DBRM into package MYCOLLID.C_SALARY.

   The implementor is now the *function package owner.*
3. The implementor then grants the EXECUTE privilege on the user-defined function package to the definer.

   ```
   GRANT EXECUTE ON PACKAGE MYCOLLID.C_SALARY
    TO definer
   ```

   As package owner, the implementor can grant execute privileges to other users, which allows those users to execute code within the package. For example:

   ```
   GRANT EXECUTE ON PACKAGE MYCOLID.C_SALARY
         TO other_user
   ```

## Defining the user-defined function (definer role)

1. The definer executes the CREATE FUNCTION statement to define the user-defined function, salary_change, to DB2.

   ```
   CREATE FUNCTION
     SALARY_CHANGE(
       VARCHAR( 6 )
       DECIMAL( 5,2 ) )
     RETURNS
       DECIMAL( 9,2 )
     SPECIFIC schema.SALCHANGE
     LANGUAGE C
     DETERMINISTIC
     MODIFIES SQL DATA
     EXTERNAL NAME C_SALARY
     PARAMETER STYLE DB2SQL
     RETURNS NULL ON NULL CALL
     NO EXTERNAL ACTION
     NO SCRATCHPAD
     NO FINAL CALL
     ALLOW PARALLEL
     NO COLLID
     ASUTIME LIMIT 1
     STAY RESIDENT NO
     PROGRAM TYPE SUB
     WLM ENVIRONMENT WLMENV
     SECURITY DB2
     NO DBINFO;
   ```

   The definer now owns the user-defined function. The definer can execute the user-defined function package, because the user-defined function package owner, in this case the implementor, granted the EXECUTE privilege to the definer (see 127) on the package that contains the user-defined function.

2. The definer then grants the EXECUTE privilege on SALARY_CHANGE to all function invokers.

   ```
   GRANT EXECUTE ON FUNCTION SALARY_CHANGE
      TO invoker1, invoker2, invoker3, invoker4
   ```

## Using the user-defined function (invoker role)

1. The invoker codes an application program, named SALARY_ADJ. The application program contains a static SQL statement that invokes the user-defined function, SALARY_CHANGE, to give the employee a 10 percent raise if the employee is not a manager, such as in the following statement:

   ```
   EXEC SQL SELECT  FIRSTNME,
                    LASTNAME
                    SALARY_CHANGE( :hvEMPNO, 10.0 )
              INTO :hvFIRSTNME,
                   :hvLASTNAME,
                   :hvSALARY
              FROM  EMP
             WHERE  EMPNO = :hvEMPNO;
   ```

2. The invoker then precompiles, compile, link-edits, and binds the invoking application's DBRM into the *invoking package* or *plan* (the package or plan that contains the SQL that invokes the user-defined function). The invoker is now the owner of the invoking plan or package.

The invoker must hold the SELECT privilege on the table EMP in addition to the EXECUTE privilege on the function SALARY_CHANGE.

### How DB2 determines authorization IDs

DB2 determines the authorization ID (invoker) that executes a user-defined function package based on whether the SQL statement that invokes the user-defined function is static or dynamic. In this example, the invoking package SALARY_ADJ contains a static SQL SELECT statement that invokes the user-defined function SALARY_CHANGE. Therefore, DB2 uses the rules for static SQL to determine the authorization ID (invoker) that executes the user-defined function package C_SALARY.

- While execution occurs in invoking package SALARY_ADJ, DB2 uses the authorization ID of the invoker, the package owner.

  The invoker requires the EXECUTE privilege on the user-defined function, SALARY_CHANGE, which the package SALARY_ADJ invokes. The user-defined function definer has the EXECUTE privilege on the user-defined function package C_SALARY, therefore, the invoker does not require the EXECUTE privilege.
- When execution changes to the user-defined function package C_SALARY, DB2 uses the authorization ID of the implementor, the package owner. The package owner is the authorization ID with authority to execute all static SQL in the user-defined function package C_SALARY.

For an example of determining authorization IDs for dynamic SQL, see "Example of determining authorization IDs for dynamic SQL statements in routines" on page 135.

## Which IDs can exercise which privileges

When a process gains access to DB2, it has a primary authorization ID, an SQL ID, and perhaps one or more secondary authorization IDs. A plan or package also has an owner ID. A specific one of those IDs must hold the required privileges for some actions; for other actions, any one or several of the IDs must hold the required privileges. Table 36 summarizes, for different actions, which IDs can provide the necessary privileges. For more specific details on any statement or command, see *DB2 SQL Reference* or *DB2 Command Reference*.

**Performance hints:** A process can have up to 245 secondary IDs. For some actions, DB2 searches a catalog table for each ID until it finds a required privilege. Therefore, the more secondary IDs that must be checked, the longer the check takes. For dynamic SQL, the current SQL ID is checked first; the operation is fastest if that ID has all the necessary privileges.

*Table 36. Required privileges for basic operations*

| Operation | ID | Required privileges |
|---|---|---|
| | **Dynamic SQL statements** | |
| GRANT | Current SQL ID | Any of these:<br>• The applicable privilege with the grant option<br>• An authority that includes the privilege, with the grant option (not needed for SYSADM or SYSCTRL)<br>• Ownership that implicitly includes the privilege |
| REVOKE | Current SQL ID | Must either have granted the privilege that is being revoked, or hold SYSCTRL or SYSADM authority |

*Table 36. Required privileges for basic operations (continued)*

| Operation | ID | Required privileges |
|---|---|---|
| CREATE, for unqualified object name | Current SQL ID | Applicable table, database, or schema privilege. |
| Qualify name of object created | ID named as owner | Applicable table or database privilege. If the current SQL ID has SYSADM authority, the qualifier can be any ID at all, and need not have any privilege. |
| Other dynamic SQL if DYNAMICRULES uses run behavior | All primary and secondary IDs and the current SQL ID together | As required by the statement; see "Composite privileges" on page 139. Unqualified object names are qualified by the value of the special register CURRENT SQLID. See "Authorization for dynamic SQL statements" on page 132. |
| Other dynamic SQL if DYNAMICRULES uses bind behavior | Plan or package owner | As required by the statement; see "Composite privileges" on page 139. DYNAMICRULES behavior determines how unqualified object names are qualified; see "Authorization for dynamic SQL statements" on page 132. |
| Other dynamic SQL if DYNAMICRULES uses define behavior | Function or procedure owner | As required by the statement; see "Composite privileges" on page 139. DYNAMICRULES behavior determines how unqualified object names are qualified; see "Authorization for dynamic SQL statements" on page 132. |
| Other dynamic SQL if DYNAMICRULES uses invoke behavior | ID of the SQL statement that invoked the function or procedure | As required by the statement; see "Composite privileges" on page 139. DYNAMICRULES behavior determines how unqualified object names are qualified; see "Authorization for dynamic SQL statements" on page 132. |
| **Operations on plans and packages** | | |
| Execute a plan | Primary or any secondary ID | Any of these:<br>• Ownership of the plan<br>• EXECUTE privilege for the plan<br>• SYSADM authority |
| Bind embedded SQL statements, for any bind operation | Plan or package owner | Any of these:<br>• Applicable privileges required by the statements<br>• Authorities that include the privileges<br>• Ownership that implicitly includes the privileges<br>Object names include the value of QUALIFIER, where it applies. |
| Include package in PKLIST[1] | Plan owner | Any of these:<br>• Ownership of the package<br>• EXECUTE privilege for the package<br>• PACKADM authority over the package collection<br>• SYSADM authority |

*Table 36. Required privileges for basic operations (continued)*

| Operation | ID | Required privileges |
|---|---|---|
| BIND a new plan using the default owner or primary authorization ID | Primary ID | BINDADD privilege, or SYSCTRL or SYSADM authority |
| BIND a new package using the default owner or primary authorization ID | Primary ID | If the value of the field BIND NEW PACKAGE on installation panel DSNTIPP is BIND, any of these:<br>• BINDADD privilege and CREATE IN privilege for the collection<br>• PACKADM authority for the collection<br>• SYSADM or SYSCTRL authority<br>If BIND NEW PACKAGE is BINDADD, any of these:<br>• BINDADD privilege and either the CREATE IN or PACKADM privilege for the collection<br>• SYSADM or SYSCTRL authority |
| BIND REPLACE or REBIND for a plan or package using the default owner or primary authorization ID | Primary or any secondary ID | Any of these:<br>• Ownership of the plan or package<br>• BIND privilege for the plan or package<br>• BINDAGENT from the plan or package owner<br>• PACKADM authority for the collection (for a package only)<br>• SYSADM or SYSCTRL authority<br>See also “Multiple actions in one statement” on page 139. |
| BIND a new version of a package, with default owner | Primary ID | If BIND NEW PACKAGE is BIND, any of these:<br>• BIND privilege on the package or collection<br>• BINDADD privilege and CREATE IN privilege for the collection<br>• PACKADM authority for the collection<br>• SYSADM or SYSCTRL authority<br>If BIND NEW PACKAGE is BINDADD, any of these:<br>• BINDADD privilege and either the CREATE IN or PACKADM privilege for the collection<br>• SYSADM or SYSCTRL authority |
| FREE or DROP a package[2] | Primary or any secondary ID | Any of these:<br>• Ownership of the package<br>• BINDAGENT from the package owner<br>• PACKADM authority for the collection<br>• SYSADM or SYSCTRL authority |
| COPY a package | Primary or any secondary ID | Any of these:<br>• Ownership of the package<br>• COPY privilege for the package<br>• BINDAGENT from the package owner<br>• PACKADM authority for the collection<br>• SYSADM or SYSCTRL authority |

*Table 36. Required privileges for basic operations (continued)*

| Operation | ID | Required privileges |
|---|---|---|
| FREE a plan | Primary or any secondary ID | Any of these:<br>• Ownership of the plan<br>• BIND privilege for the plan<br>• BINDAGENT from the plan owner<br>• SYSADM or SYSCTRL authority |
| Name a new OWNER other than the primary authorization ID for any bind operation | Primary or any secondary ID | Any of these:<br>• New owner is the primary or any secondary ID<br>• BINDAGENT from the new owner<br>• SYSADM or SYSCTRL authority |

**Notes:**

1. A user-defined function, stored procedure, or trigger package does not need to be included in a package list.
2. A trigger package cannot be deleted by FREE PACKAGE or DROP PACKAGE. The DROP TRIGGER statement must be used to delete the trigger package.

## Authorization for dynamic SQL statements

This section explains authorization behavior for dynamic SQL statements. The two key factors that influence authorization behavior are the DYNAMICRULES value and the run time environment of a package.

The BIND or REBIND option DYNAMICRULES determines what values apply at run time for the following dynamic SQL attributes:

- The authorization ID that is used to check authorization
- The qualifier that is used for unqualified objects
- The source for application programming options that DB2 uses to parse and semantically verify dynamic SQL statements
- Whether dynamic SQL statements can include GRANT, REVOKE, ALTER, CREATE, DROP, and RENAME statements

In addition to the DYNAMICRULES value, the run-time environment of a package controls how dynamic SQL statements behave at run time. The two possible run-time environments are:

- The package runs as part of a stand-alone program.
- The package runs as a stored procedure or user-defined function package, or runs under a stored procedure or user-defined function.

  A package that runs under a stored procedure or user-defined function is a package whose associated program meets one of the following conditions:

  – The program is called by a stored procedure or user-defined function.

  – The program is in a series of nested calls that start with a stored procedure or user-defined function.

The combination of the DYNAMICRULES value and the run-time environment determine the values for the dynamic SQL attributes. That set of attribute values is called the dynamic SQL statement *behavior*. The four behaviors are:

- Run behavior
- Bind behavior
- Define behavior
- Invoke behavior

This section explains each behavior. The behaviors are summarized in Table 38 on page 135 . The DYNAMICRULES options associated with each behavior are summarized in Table 37 on page 134.

## Run behavior

DB2 processes dynamic SQL statements using the standard attribute values for dynamic SQL statements, which are collectively called *run behavior*:

- DB2 uses the authorization ID of the application process and the SQL authorization ID (the value of the CURRENT SQLID special register):
  - For authorization checking of dynamic SQL statements
  - As the implicit qualifier of table, view, index, and alias names
- Dynamic SQL statements use the values of application programming options that were specified during installation. The installation option USE FOR DYNAMICRULES has no effect.
- GRANT, REVOKE, CREATE, ALTER, DROP, and RENAME statements can be executed dynamically.

## Bind behavior

DB2 processes dynamic SQL statements using the following attribute values, which are collectively called *bind behavior*:

- DB2 uses the authorization ID of the plan or package for authorization checking of dynamic SQL statements.
- Unqualified table, view, index, and alias names in dynamic SQL statements are implicitly qualified with value of the bind option QUALIFIER; if you do not specify QUALIFIER, DB2 uses the authorization ID of the plan or package owner as the implicit qualifier.
- The attribute values that are described in “Common attribute values for bind, define, and invoke behavior” on page 134.

The values of the authorization ID and the qualifier for unqualified objects are the same as those that are used for embedded or static SQL statements.

## Define behavior

When the package is run as or under a stored procedure or user-defined function package or runs under a stored procedure or user-defined function, DB2 processes dynamic SQL statements using *define* behavior, which consists of the following attribute values:

- DB2 uses the authorization ID of the user-defined function or stored procedure owner for authorization checking of dynamic SQL statements in the application package.
- The default qualifier for unqualified objects is the user-defined function or stored procedure owner.
- The attribute values that are described in “Common attribute values for bind, define, and invoke behavior” on page 134.

When the package is run as a stand-alone program, DB2 processes dynamic SQL statements using bind behavior or run behavior, depending on the DYNAMICRULES value specified.

## Invoke behavior

When the package is run as or under a stored procedure or user-defined function package or runs under a stored procedure or user-defined function, DB2 processes dynamic SQL statements using *invoke* behavior, which consists of the following attribute values:

- DB2 uses the authorization ID of the user-defined function or stored procedure invoker for authorization checking of dynamic SQL statements in the application package.

  If the invoker is the primary authorization ID of the process or the CURRENT SQLID value, secondary authorization IDs will also be checked if they are needed for the required authorization. Otherwise, only one ID, the ID of the invoker, is checked for the required authorization.
- The default qualifier for unqualified objects is the user-defined function or stored procedure invoker.
- The attribute values that are described in “Common attribute values for bind, define, and invoke behavior”.

When the package is run as a stand-alone program, DB2 processes dynamic SQL statements using bind behavior or run behavior, depending on the DYNAMICRULES value specified.

## Common attribute values for bind, define, and invoke behavior

The following attribute values apply to dynamic SQL statements in plans or packages that have bind, define, or invoke behavior:

- You can execute the statement SET CURRENT SQLID in a package or plan that is bound with any DYNAMICRULES value. However, DB2 does not use the value of CURRENT SQLID as the authorization ID for dynamic SQL statements.

  DB2 always uses the value of CURRENT SQLID as the qualifier for the EXPLAIN output PLAN_TABLE.
- If the value of installation option USE FOR DYNAMICRULES is YES, DB2 uses the application programming default values that were specified during installation to parse and semantically verify dynamic SQL statements. If the value of USE for DYNAMICRULES is NO, DB2 uses the precompiler options to parse and semantically verify dynamic SQL statements. For a list of the application programming defaults that USE FOR DYNAMICRULES affects, see Part 5 of *DB2 Application Programming and SQL Guide*.
- GRANT, REVOKE, CREATE, ALTER, DROP, and RENAME statements cannot be executed dynamically.

Table 37 shows the combination of DYNAMICRULES value and run-time environment that yield each dynamic SQL behavior. Table 38 on page 135 shows the dynamic SQL attribute values for each type of dynamic SQL behavior.

*Table 37. How DYNAMICRULES and the run-time environment determine dynamic SQL statement behavior*

| | **Behavior of dynamic SQL statements** | |
|---|---|---|
| **DYNAMICRULES value** | **Stand-alone program environment** | **User-defined function or stored procedure environment** |
| BIND | Bind behavior | Bind behavior |
| RUN | Run behavior | Run behavior |
| DEFINEBIND | Bind behavior | Define behavior |
| DEFINERUN | Run behavior | Define behavior |
| INVOKEBIND | Bind behavior | Invoke behavior |
| INVOKERUN | Run behavior | Invoke behavior |

**Notes:**

1. The BIND and RUN values can be specified for packages and plans. The other values can be specified only for packages.

*Table 38. Definitions of dynamic SQL statement behaviors*

| | Setting for dynamic SQL attributes | | | |
|---|---|---|---|---|
| **Dynamic SQL attribute** | **Bind behavior** | **Run behavior** | **Define behavior** | **Invoke behavior** |
| Authorization ID | Plan or package owner | Current SQLID | User-defined function or stored procedure owner | Authorization ID of invoker [1] |
| Default qualifier for unqualified objects | Bind OWNER or QUALIFIER value | Current SQLID | User-defined function or stored procedure owner | Authorization ID of invoker |
| CURRENT SQLID [2] | Not applicable | Applies | Not applicable | Not applicable |
| Source for application programming options | Determined by DSNHDECP parameter DYNRULS [3] | Install panel DSNTIPF | Determined by DSNHDECP parameter DYNRULS [3] | Determined by DSNHDECP parameter DYNRULS [3] |
| Can execute GRANT, REVOKE, CREATE, ALTER, DROP, RENAME? | No | Yes | No | No |

**Notes:**

1. If the invoker is the primary authorization ID of the process or the CURRENT SQLID value, secondary authorization IDs will also be checked if they are needed for the required authorization. Otherwise, only one ID, the ID of the invoker, is checked for the required authorization.
2. DB2 uses the value of CURRENT SQLID as the authorization ID for dynamic SQL statements only for plans and packages that have DYNAMICRULES run behavior. For the other dynamic SQL behaviors, DB2 uses the authorization ID that is associated with each dynamic SQL behavior, as shown in this table.

   The value to which CURRENT SQLID is initialized is independent of the dynamic SQL behavior. For stand-alone programs, CURRENT SQLID is initialized to the primary authorization ID. See *DB2 Application Programming and SQL Guide* for information on initialization of CURRENT SQLID for user-defined functions and stored procedures.

   You can execute the SET CURRENT SQLID statement to change the value of CURRENT SQLID for packages with any dynamic SQL behavior, but DB2 uses the CURRENT SQLID value only for plans and packages with run behavior.
3. The value of DSNHDECP parameter DYNRULS, which you specify in field USE FOR DYNAMICRULES in installation panel DSNTIPF, determines whether DB2 uses the precompiler options or the application programming defaults for dynamic SQL statements. See Part 5 of *DB2 Application Programming and SQL Guide* for more information.

## Example of determining authorization IDs for dynamic SQL statements in routines

Suppose that A is a stored procedure and C is a program that is neither a user-defined function nor a stored procedure. Also suppose that subroutine B is called by both stored procedure A and program C. Subroutine B, which is invoked by a language call, is neither a user-defined function nor a stored procedure. AP is the package that is associated with stored procedure A, and BP is the package that is associated with subroutine B. A, B, and C execute as shown in Figure 10 .

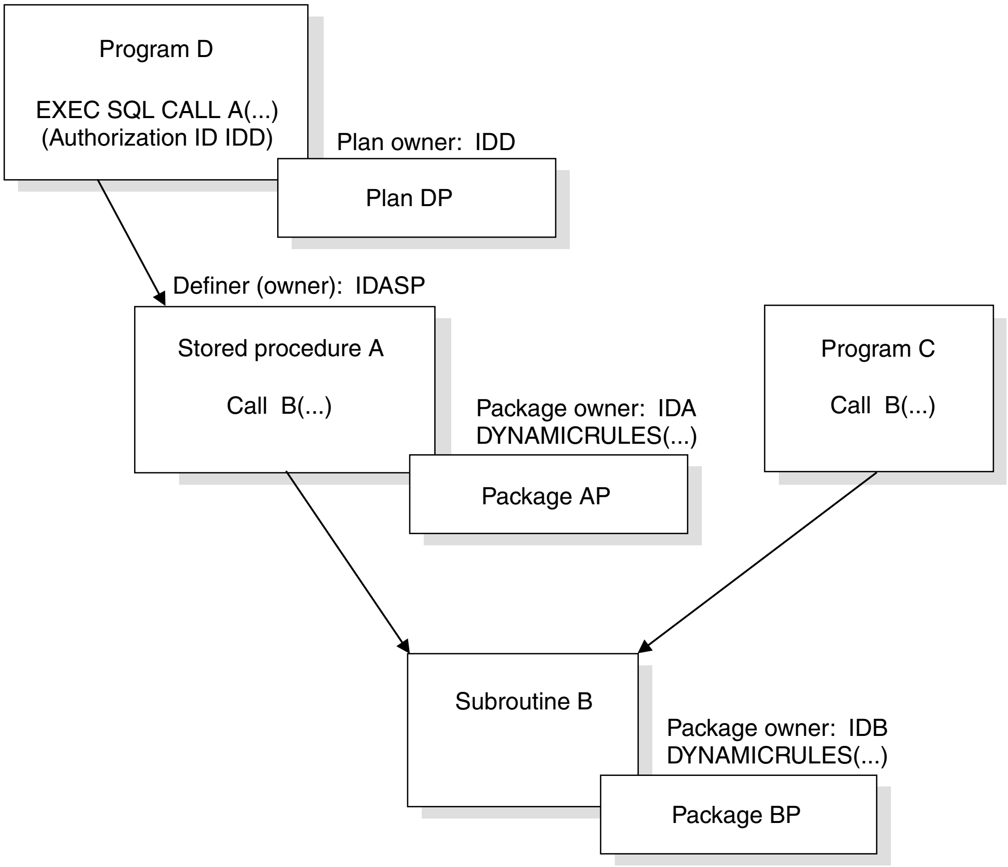


*Figure 10. Authorization for dynamic SQL statements in programs and routines*

Stored procedure A was defined by IDASP and is therefore owned by IDASP. The stored procedure package AP was bound by IDA and is therefore owned by IDA. Package BP was bound by IDB and is therefore owned by IDB. The authorization ID under which EXEC SQL CALL A runs is IDD, the owner of plan DP.

The authorization ID under which dynamic SQL statements in package AP run is determined in the following way:

- If package AP uses DYNAMICRULES bind behavior, the authorization ID for dynamic SQL statements in package AP is IDA, the owner of package AP.
- If package AP uses DYNAMICRULES run behavior, the authorization ID for dynamic SQL statements in package AP is the value of CURRENT SQLID when the statements execute.
- If package AP uses DYNAMICRULES define behavior, the authorization ID for dynamic SQL statements in package AP is IDASP, the definer (owner) of stored procedure A.
- If package AP uses DYNAMICRULES invoke behavior, the authorization ID for dynamic SQL statements in package AP is IDD, the invoker of stored procedure A.

The authorization ID under which dynamic SQL statements in package BP run is determined in the following way:

- If package BP uses DYNAMICRULES bind behavior, the authorization ID for dynamic SQL statements in package BP is IDB, the owner of package BP.
- If package BP uses DYNAMICRULES run behavior, the authorization ID for dynamic SQL statements in package BP is the value of CURRENT SQLID when the statements execute.
- If package BP uses DYNAMICRULES define behavior:
  - When subroutine B is called by stored procedure A, the authorization ID for dynamic SQL statements in package BP is IDASP, the definer of stored procedure A.
  - When subroutine B is called by program C:
    - If package BP uses the DYNAMICRULES option DEFINERUN, DB2 executes package BP using DYNAMICRULES run behavior, which means that the authorization ID for dynamic SQL statements in package BP is the value of CURRENT SQLID when the statements execute.
    - If package BP uses the DYNAMICRULES option DEFINEBIND, DB2 executes package BP using DYNAMICRULES bind behavior, which means that the authorization ID for dynamic SQL statements in package BP is IDB, the owner of package BP.
- If package BP uses DYNAMICRULES invoke behavior:
  - When subroutine B is called by stored procedure A, the authorization ID for dynamic SQL statements in package BP is IDD, the authorization ID under which EXEC SQL CALL A executed.
  - When subroutine B is called by program C:
    - If package BP uses the DYNAMICRULES option INVOKERUN, DB2 executes package BP using DYNAMICRULES run behavior, which means that the authorization ID for dynamic SQL statements in package BP is the value of CURRENT SQLID when the statements execute.
    - If package BP uses the DYNAMICRULES option INVOKEBIND, DB2 executes package BP using DYNAMICRULES bind behavior, which means that the authorization ID for dynamic SQL statements in package BP is IDB, the owner of package BP.

Now suppose that B is a user-defined function, as shown in Figure 11 on page 138.

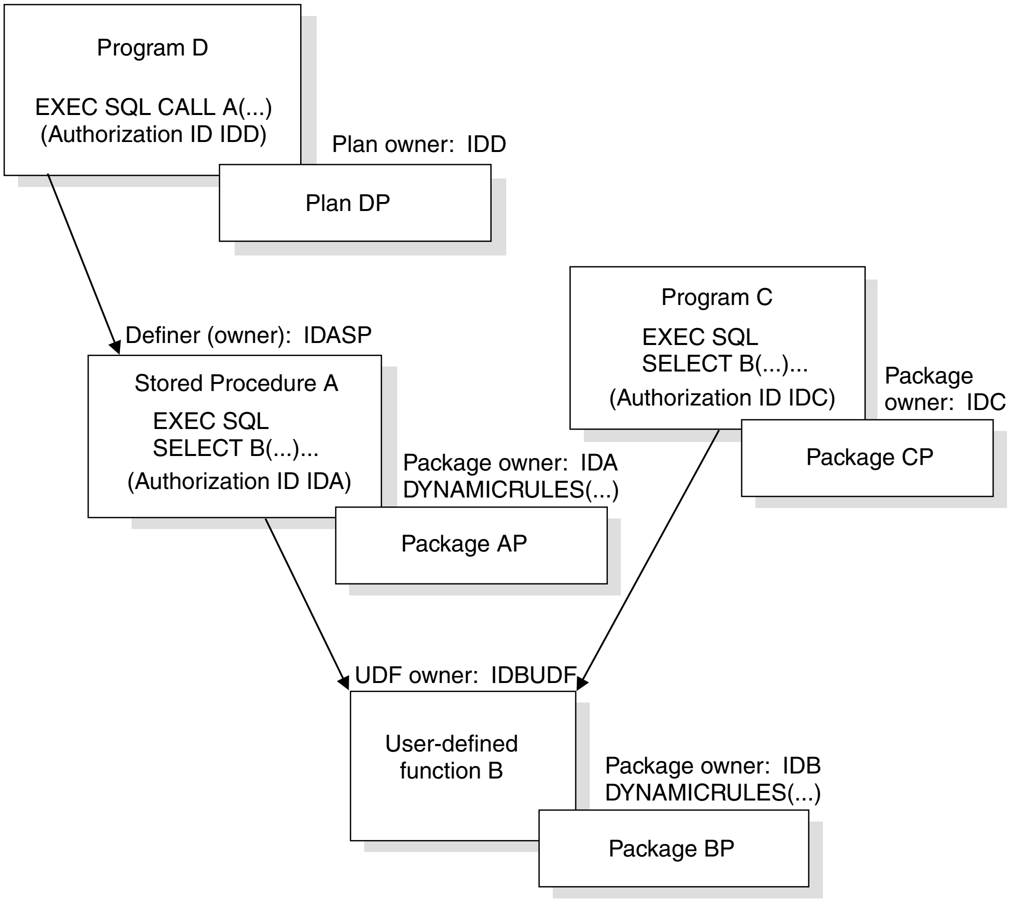


*Figure 11. Authorization for dynamic SQL statements in programs and nested routines*

User-defined function B was defined by IDBUDF and is therefore owned by ID IDBUDF. Stored procedure A invokes user-defined function B under authorization ID IDA. Program C invokes user-defined function B under authorization ID IDC. In both cases, the invoking SQL statement (EXEC SQL SELECT B) is static.

The authorization ID under which dynamic SQL statements in package BP run is determined in the following way:

- If package BP uses DYNAMICRULES bind behavior, the authorization ID for dynamic SQL statements in package BP is IDB, the owner of package BP.
- If package BP uses DYNAMICRULES run behavior, the authorization ID for dynamic SQL statements in package BP is the value of CURRENT SQLID when the statements execute.
- If package BP uses DYNAMICRULES define behavior, the authorization ID for dynamic SQL statements in package BP is IDBUDF, the definer of user-defined function B.
- If package BP uses DYNAMICRULES invoke behavior:
  - When user-defined function B is invoked by stored procedure A, the authorization ID for dynamic SQL statements in package BP is IDA, the authorization ID under which B is invoked in stored procedure A.
  - When user-defined function B is invoked by program C, the authorization ID for dynamic SQL statements in package BP is IDC, the owner of package CP, and is the authorization ID under which B is invoked in program C.

### Simplifying authorization

You can simplify authorization in several ways. Make sure you do not violate any of the authorization standards at your installation:

- Have the implementor bind the user-defined function package using DYNAMICRULES define behavior. With this behavior in effect, DB2 only needs to check one ID to execute dynamic SQL statements in the routine, the definer's, rather than check the many different IDs that invoke the user-defined function.
- If you have many different routines, group those routines into schemas. Then, grant EXECUTE on the routines in the schema to the appropriate users. Users have execute authority on any functions you add to that schema. For example:

  ```
  GRANT EXECUTE ON FUNCTION schemaname.* TO PUBLIC;
  ```

## Composite privileges

An SQL statement can name more than one object; for example, a SELECT operation can join two or more tables, or an INSERT can use a subquery. Those operations require privileges on all the tables. You might be able to issue such a statement dynamically even though one of your IDs alone does not have all the required privileges.

If DYNAMICRULES run behavior is in effect when the dynamic statement is prepared, it is validated if the set of your primary and all your secondary IDs has all the needed privileges among them. If you embed the same statement in a host program and try to bind it into a plan or package, the validation fails. The validation also fails for the dynamic statement if DYNAMICRULES bind, define, or invoke behavior is in effect when you issue the dynamic statement. In each case, all the required privileges must be held by the single authorization ID, determined by DYNAMICRULES behavior.

## Multiple actions in one statement

A REBIND or FREE command can name more than one plan or package. If no owner is named, the set of privileges associated with the primary and secondary IDs must include the BIND privilege for each object. For example, suppose that FREDDY has the BIND privilege on plan P1 and that REUBEN has the BIND privilege on plan P2. Assume someone with FREDDY and REUBEN as secondary authorization IDs issues the following command:

```
REBIND PLAN(P1,P2)
```

P1 and P2 are successfully rebound, even though neither FREDDY nor REUBEN has the BIND privilege for both plans.

## Some role models

The names of some authorities suggest job titles. For example, you might expect a system administrator to have SYSADM authority. But not all organizations divide job responsibilities in the same way. The table below lists some other common job titles, the tasks that usually go with them, and the DB2 authorities or privileges that are needed to carry out those tasks.

*Table 39. Some common jobs, tasks, and required privileges*

| Job title | Tasks | Required privileges |
|---|---|---|
| System Operator | Issues commands to start and stop DB2; control traces; display databases and threads; recover indoubt threads; start, stop, and display routines. | SYSOPR authority. |

*Table 39. Some common jobs, tasks, and required privileges (continued)*

| Job title | Tasks | Required privileges |
|---|---|---|
| System Administrator | Performs emergency backup, with access to all data. | SYSADM authority. |
| Security Administrator | Authorizes other users, for some or all levels below. | SYSCTRL authority. |
| Database Administrator | Designs, creates, loads, reorganizes, and monitors databases, tables, and other objects. | DBADM authority over a database; use of storage groups and buffer pools. |
| System Programmer | Installs a DB2 subsystem; recovers the DB2 catalog; repairs data. | Installation SYSADM, which is assigned when DB2 is installed. (Consider securing the password for an ID with this authority so that the authority is available only when needed.) |
| Application Programmer | Develops and tests DB2 application programs; can create tables of test data. | BIND on existing plans or packages, or BINDADD; CREATE IN on some collections; privileges on some objects; CREATETAB on some database, with a default table space provided. |
| Production Binder | Binds, rebinds, and frees application plans. | BINDAGENT, granted by users with BINDADD and CREATE IN privileges. |
| Package Administrator | Manages collections and the packages in them, and delegates the responsibilities. | PACKADM authority. |
| User Analyst | Defines the data requirements for an application program, by examining the DB2 catalog. | SELECT on the SYSTABLES, SYSCOLUMNS, and SYSVIEWS catalog tables. CREATETMTAB system privilege to create created temporary tables. |
| Program End User | Executes an application program. | EXECUTE for the application plan. |
| Information Center Consultant | Defines the data requirements for a query user; provides the data by creating tables and views, loading tables, and granting access. | DBADM authority over some database; SELECT on the SYSTABLES, SYSCOLUMNS, and SYSVIEWS catalog tables. |
| Query User | Issues SQL statements to retrieve, add, or change data. Can save results as tables or in global temporary tables. | SELECT, INSERT, UPDATE, DELETE on some tables and views; CREATETAB, to create tables in other than the default database; CREATETMTAB system privilege to create temporary tables; SELECT on SYSTABLES, SYSCOLUMNS, or views thereof. QMF provides the views. |

## Examples of granting and revoking privileges

The SQL GRANT statement lets you grant privileges explicitly. The REVOKE statement lets you take them away. Only a privilege that has been specifically granted can be revoked. (You can use either statement only if authorization checking was enabled when DB2 was installed.)

You can grant and revoke privileges to and from a single ID, or you can name several IDs on one statement. You can grant privileges to the ID PUBLIC, making them available to all IDs at the local DB2, including the owner IDs of packages that are bound from a remote location.

When you grant any privilege to PUBLIC, DB2 catalog tables record the grantee of the privilege as PUBLIC. Implicit table privileges are also granted to PUBLIC for declared temporary tables. PUBLIC is a special identifier used by DB2 internally; do not use PUBLIC as a primary or secondary authorization ID. When a privilege is revoked from PUBLIC, authorization IDs to which the privilege was specifically granted still retain the privilege.

The holding of other privileges can depend on privileges granted to PUBLIC. Then, GRANTOR is listed as PUBLIC, as in the following examples:

- USER1 creates a table and grants ALL PRIVILEGES on it to PUBLIC. USER2 then creates a view on the table. In the catalog table SYSIBM.SYSTABAUTH, GRANTOR is PUBLIC and GRANTEE is USER2. Creating the view requires the SELECT privilege, which is held by PUBLIC. If PUBLIC loses the privilege, the view is dropped.
- Another user binds a plan, PLAN1, whose program refers to the table that was created in the previous example. In SYSTABAUTH, GRANTOR is PUBLIC, GRANTEE is PLAN1, and GRANTEETYPE is P. Again, if PUBLIC loses its privilege, the plan can be invalidated.

You can grant a specific privilege on one object in a single statement, you can grant a list of privileges, and you can grant privileges over a list of objects. You can also grant ALL, for all the privileges of accessing a single table, or for all privileges that are associated with a specific package. If the same grantor grants access to the same grantee more than once, without revoking it, DB2 ignores the duplicate grants and keeps only one record in the catalog for the authorization. That suppression of duplicate records applies not only to explicit grants, but also to the implicit grants of privileges that are made when a package is created.

***Granting privileges to remote users:*** A query that arrives at your local DB2 through the distributed data facility is accompanied by an authorization ID. That ID can go through connection or sign-on processing when it arrives, can be translated to another value, and can be associated with secondary authorization IDs. (For the details of all those processes, see “Controlling requests from remote applications” on page 176.)

The end result is that the query is associated with a set of IDs that is known to your local DB2. How you assign privileges to those IDs is no different from how you assign them to IDs that are associated with local queries.

You can grant a table privilege to any ID anywhere that uses DB2 private protocol access to your data, by issuing:

```
GRANT privilege TO PUBLIC AT ALL LOCATIONS;
```

The privilege can be any table privilege except ALTER, INDEX, REFERENCES, or TRIGGER.

If you grant to PUBLIC AT ALL LOCATIONS, the grantee is PUBLIC*. PUBLIC is a special identifier used by DB2 internally; do not use PUBLIC* as a primary or secondary authorization ID. When a privilege is revoked from PUBLIC AT ALL LOCATIONS, authorization IDs to which the privilege was specifically granted still retain the privilege.

There are, however, some differences in the privileges that a query using DB2 private protocol access can use:

- It cannot use privileges granted TO PUBLIC; it can use privileges granted TO PUBLIC AT ALL LOCATIONS.
- It can exercise only the SELECT, INSERT, UPDATE, and DELETE privileges at the remote location.

Those restrictions do not apply to queries run by a package bound at your local DB2. Those queries can use any privilege granted to their associated IDs or any privilege granted to PUBLIC.

# Examples using GRANT

The scenario in this section illustrates the different types of grant statements. The data in this scenario is not highly critical, because the focus is on GRANT rather than on the broader topic of security.

Suppose that the Spiffy Computer Company wants to create a database to hold information that is usually posted on hallway bulletin boards—things like notices of upcoming holidays and bowling scores. The president of the Spiffy Computer Company, Truly Spiffy, is a wonderful bowler with a great ego, and wants everyone in the company to have access to her scores.

To create and maintain the tables and programs that are needed for this application, the security plan provides for the roles shown in Figure 12.

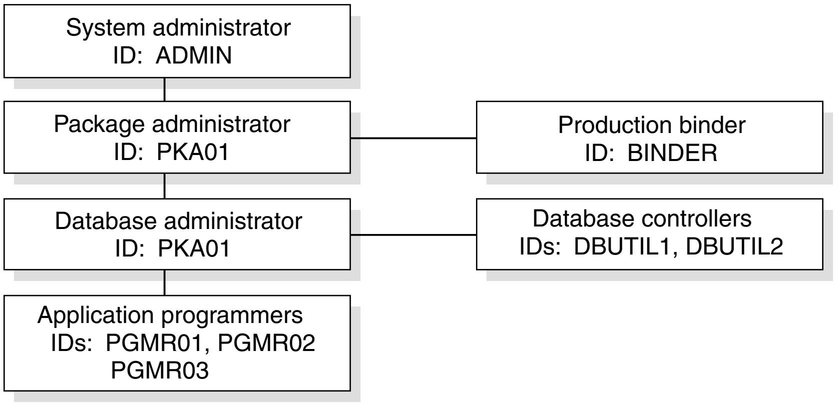


*Figure 12. Security plan for the Spiffy Computer Company. Lines connect the grantor of a privilege or authority to the grantee.*

Spiffy's system of privileges and authorities associates each role with an authorization ID.

## System administrator's privileges

User ID: ADMIN
Authority: SYSADM
Privileges: Ownership of SG1

The system administrator uses the ADMIN authorization ID, which has SYSADM authority, to create a storage group (SG1) and issue the following statements:

1. `GRANT PACKADM ON COLLECTION BOWLS TO PKA01 WITH GRANT OPTION;`

   This grants package privileges on all packages in the collection BOWLS, plus the CREATE IN privilege on that collection to PKA01, who can also grant those privileges to others.
2. `GRANT CREATEDBA TO DBA01;`

This grants the privilege to create a database and have DBADM authority over it to DBA01.

3. `GRANT USE OF STOGROUP SG1 TO DBA01 WITH GRANT OPTION;`

   This allows DBA01 to use storage group SG1 and to grant that privilege to others.

4. `GRANT USE OF BUFFERPOOL BP0, BP1 TO DBA01 WITH GRANT OPTION;`

   This allows DBA01 to use buffer pools BP0 and BP1 and to grant that privilege to others.

### Package administrator's privileges

User ID: PKA01
Authority: PACKADM over the collection BOWLS

The package administrator, PKA01, controls the binding of packages into collections and can grant the CREATE IN privilege and the package privileges to others.

### Database administrator's privileges

User ID: DBA01
Authority: DBADM over DB1
Privileges: CREATEDBA
Use of SG1 with GRANT
Use of BP0 and BP1 with GRANT
Ownership of DB1

The database administrator, DBA01, using the CREATEDBA privilege, creates the database DB1. Then DBA01 automatically has DBADM authority over the database.

### Database controller's privileges

User ID: DBUTIL1, DBUTIL2
Authority: DBCTRL over DB1

The database administrator at Spiffy wants help running the COPY and RECOVER utilities and therefore grants DBCTRL authority over database DB1 to DBUTIL1 and DBUTIL2.

To do that, the database administrator issues the following statement:

```
GRANT DBCTRL ON DATABASE DB1 TO DBUTIL1, DBUTIL2;
```

## Examples with secondary IDs

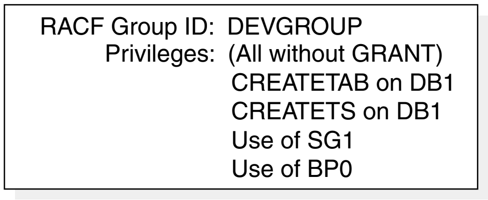
The examples that follow illustrate the use of secondary authorization IDs.

That means using RACF (or a similar external security system) to define user groups and connect primary authorization IDs to them. The primary DB2 authorization ID is the user's RACF user ID, and the associated secondary authorization IDs are the names of the groups to which the primary ID is connected. DB2 privileges are then granted to the secondary IDs but might not be explicitly granted to any primary ID.

This approach reduces the number of grants that are needed and associates privileges with a *functional* ID, rather than an *individual* one. The functional ID can remain in place until Spiffy redesigns its procedures. Individual IDs, which come and go, can be connected to or disconnected from the group that exercises the functional ID's privileges, without requiring new grants or revokes.

## Application programmers' privileges

The database administrator at Spiffy wants several employees in the Software Support department to create tables in the DB1 database and creates DEVGROUP as a RACF group ID for this purpose. To make things simpler, the database administrator decides that each CREATE TABLE statement should implicitly create a unique table space for the table. Hence, DEVGROUP needs the CREATETAB and CREATETS privileges, and the privileges to use the SG1 storage group and one of the buffer pools, BP0, for the implicitly created table spaces. The following figure shows this group and their privileges:



```
RACF Group ID:  DEVGROUP
     Privileges:  (All without GRANT)
                  CREATETAB on DB1
                  CREATETS on DB1
                  Use of SG1
                  Use of BP0
```

The database administrator, DBA01, owns database DB1 and has the privileges to use storage group SG1 and buffer pool BP0 (both with the GRANT option). The database administrator issues the following statements:

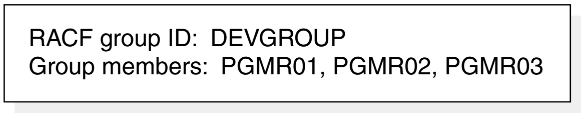
1. `GRANT CREATETAB, CREATETS ON DATABASE DB1 TO DEVGROUP;`
2. `GRANT USE OF STOGROUP SG1 TO DEVGROUP;`
3. `GRANT USE OF BUFFERPOOL BP0 TO DEVGROUP;`

The system and database administrators at Spiffy still need to control the use of those resources, so the statements above are issued without the GRANT option.

Three programmers in the Software Support department write and test a new program, PROGRAM1. Their IDs are PGMR01, PGMR02, and PGMR03. Each one needs to create test tables, use the SG1 storage group, and use one of the buffer pools. However, all of those resources are controlled by DEVGROUP, which is a RACF group ID.

Therefore, granting privileges over those resources specifically to PGMR01, PGMR02, and PGMR03 is unnecessary. All that is needed is to connect each ID to the RACF group DEVGROUP. (Assuming that the installed connection and sign-on procedures allow secondary authorization IDs. For examples of RACF commands that connect IDs to RACF groups, and for a description of the connection and sign-on procedures, see “Chapter 12. Controlling access to a DB2 subsystem” on page 169.)

The following figure shows this group and its members:



RACF group ID:  DEVGROUP
Group members:  PGMR01, PGMR02, PGMR03

The security administrator connects as many members as desired to the group DEVGROUP. Each member can exercise all the privileges that are granted to the group ID.

## Privileges for binding the plan

Three programmers can now share the tasks done by the ID DEVGROUP. Someone creates a test table, DEVGROUP.T1, in database DB1 and loads it with test data. Someone writes a program, PROGRAM1, to display bowling scores that are contained in T1. Someone must bind the plan and packages that accompany the program, and that requires an additional privilege. The following figure shows the BINDADD privilege granted to the group:

RACF group ID: DEVGROUP
Privilege: BINDADD

ADMIN, who has SYSADM authority, grants the required privilege by issuing the following statement:

```
GRANT BINDADD TO DEVGROUP;
```

With that privilege, any member of the RACF group DEVGROUP can bind plans and packages that are to be owned by DEVGROUP. Any member of the group can rebind a plan or package that is owned by DEVGROUP.

The Software Support department proceeds to create and test the program.

## Moving PROGRAM1 into production

Spiffy has a different set of tables, containing actual data that is owned by another group ID, PRODCTN. The program was written with unqualified table names; the new packages and plan must refer to table PRODCTN.T1. To move the completed program into production, someone must:

- Rebind the application plan with the owner PRODCTN.
- Rebind the packages into the collection BOWLS, again with the owner PRODCTN.

Spiffy gives that job to a production binder, with the ID BINDER. BINDER needs privileges to bind a plan or package that DEVGROUP owns, to bind a plan or package with OWNER (PRODCTN), and to add a package to the collection BOWLS. The following figure shows the privileges for BINDER:

User ID: BINDER
Privileges: BINDAGENT for DEVGROUP
BINDAGENT for PRODCTN
CREATE on BOWLS

Any member of the group DEVGROUP can grant the BINDAGENT privilege, by using the statements below. Any member of PRODCTN can also grant the BINDAGENT privilege, by using a similar set of statements.

1. `SET CURRENT SQLID='DEVGROUP';`
2. `GRANT BINDAGENT TO BINDER;`

The package administrator for BOWLS, PACKADM, can grant the CREATE privilege with this statement:

```
GRANT CREATE ON COLLECTION BOWLS TO BINDER;
```

With the plan in place, the database administrator at Spiffy wants to make the PROGRAM1 plan available to all employees by issuing the statement:

```
GRANT EXECUTE ON PLAN PROGRAM1 TO PUBLIC;
```

More than one ID has the authority or privileges necessary to issue this statement. ADMIN has SYSADM authority and can grant the EXECUTE privilege. Or, PGMR01 can set CURRENT SQLID to PRODCTN, which owns PROGRAM1, and issue the statement. When EXECUTE is granted to public, other IDs do not need any explicit authority on T1; having the privilege of executing the plan is sufficient.

Finally, the plan to display bowling scores at Spiffy Computer Company is complete. The production plan, PROGRAM1, is created, and all IDs have the authority to execute the plan.

### Spiffy's approach to distributed data

Some time after the system and database administrators at Spiffy install their security plan, Truly Spiffy tells them that other applications on other systems must connect to the local DB2. She wants people at every location to be able to access bowling scores through PROGRAM1 on the local system.

The solution is to:

1. Add a CONNECT statement to the program, naming the location at which table PRODCTN.T1 resides. (In this case, the table and the package reside at only the central location.)
2. Issue the statement: `GRANT CREATE IN COLLECTION BOWLS TO DEVGROUP;` (PKA01, who has PACKADM authority, grants the required privileges to DEVGROUP by issuing this statement.)
3. Bind the SQL statements in PROGRAM1 as a package.

After that is done, the package owner can issue the statement:

```
GRANT EXECUTE ON PACKAGE PROGRAM1 TO PUBLIC;
```

Any system that is connected to the original DB2 location can then run PROGRAM1 and execute the package, using DRDA access. (If the remote system is another DB2, a plan must be bound there that includes the package in its package list.)

That solution, of course, is vastly simplified. Here the focus is on granting appropriate privileges and authorities. In practice, you would also need to consider questions like these:

- Is the performance of a remote query acceptable for this application?
- If other DBMSs are not DB2 subsystems, will the non-SQL portions of PROGRAM1 run in their environments?

## The REVOKE statement

An ID that has granted a privilege can revoke it by issuing the REVOKE statement:

```
REVOKE authorization-specification FROM auth-id
```

An ID with SYSADM or SYSCTRL authority can revoke a privilege that has been granted by another ID with:

```
REVOKE authorization-specification FROM auth-id BY auth-id
```

The BY clause specifies the authorization ID that originally granted the privilege. If two or more grantors grant the same privilege to an ID, executing a single REVOKE statement does not remove the privilege. To remove it, each grant of the privilege must be revoked.

The WITH GRANT OPTION clause of the GRANT statement allows an ID to pass the granted privilege to others. If the privilege is removed from the ID, its deletion can cascade to others, with side effects that are not immediately evident. When a privilege is removed from authorization ID X, it is also removed from any ID to which X granted it, unless that ID also has the privilege from some other source.[5]

For example, suppose that DBA01 has granted DBCTRL authority with the GRANT option on database DB1 to DBUTIL1, and DBUTIL1 has granted the CREATETAB privilege on DB1 to PGMR01. If DBA01 revokes DBCTRL from DBUTIL1, PGMR01 loses the CREATETAB privilege. If PGMR01 also granted that to OPER1 and OPER2, they also lose it. However, table T1, which PGMR01 created while enjoying the CREATETAB privilege, is not dropped, and the privileges that PGMR01 has or granted as its owner are not deleted. If PGMR01 granted SELECT on T1 to OPER1, the validity of that grant rests on PGMR01's ownership of the table. Even when the privilege of creating the table is revoked, the table remains, the privilege remains, and OPER1 can still access T1.

## Privileges granted from two or more IDs

In addition to the CREATETAB privilege that is granted by DBUTIL1, suppose DBUTIL2 also granted the CREATETAB privilege to PGMR01. The action is recorded in the catalog, with its date and time, but it has no other effect until the grant from DBUTIL1 to PGMR01 is revoked. Then it is necessary to determine by what authority PGMR01 granted CREATETAB to OPER1 and the others. Figure 13 diagrams the situation; arrows represent the granting of the CREATETAB privilege.

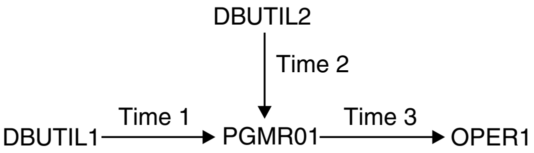


*Figure 13. Authorization granted by two or more IDs*

As in the diagram, suppose that DBUTIL1 and DBUTIL2 at Time 1 and Time 2, respectively, each issue this statement:

```
GRANT CREATETAB ON DATABASE DB1 TO PGMR01 WITH GRANT OPTION;
```

At Time 3, PGMR01 grants the privilege to OPER1. Later, DBUTIL1's authority is revoked, or perhaps DBUTIL1 explicitly revokes the CREATETAB privilege from PGMR01. PGMR01 has the privilege also from DBUTIL2, and does not lose it. Does OPER1 lose the privilege?

- If Time 3 is *later* than Time 2, OPER1 *does not* lose the privilege. The recorded dates and times show that, at Time 3, PGMR01 could have granted the privilege entirely on the basis of the privilege that was granted by DBUTIL2. That privilege was not revoked.
- If Time 3 is *earlier* than Time 2, OPER1 *does* lose the privilege. The recorded dates and times show that, at Time 3, PGMR01 could only have granted the privilege on the basis of the privilege that was granted by DBUTIL1. That privilege was revoked, so the privileges dependent on it are also revoked.

## Revoking privileges granted by other IDs

An ID with SYSADM or SYSCTRL authority can revoke privileges that are granted by other IDs.

To revoke the CREATETAB privilege on database DB1 from PGMR01 entirely, use:

[5] DB2 does not cascade a revoke of SYSADM authority from the installation SYSADM authorization IDs.

```
REVOKE CREATETAB ON DATABASE DB1 FROM PGMR01 BY ALL;
```

To revoke privileges that are granted by DBUTIL1 and to leave intact the same privileges if they were granted by any other ID, use:

```
REVOKE CREATETAB, CREATETS ON DATABASE DB1 FROM PGMR01 BY DBUTIL1;
```

### Restricting revocation of privileges

The RESTRICT clause of the REVOKE statement applies to user-defined functions, JARS (Java classes for a routine), stored procedures, and distinct types. RESTRICT must be specified to either:

- Revoke the EXECUTE privilege on a user-defined function, JAR, or stored procedure
- Revoke the USAGE privilege on a distinct type

When an attempt is made to revoke one of these privileges, DB2 determines whether the revokee owns an object that is dependent on the privilege. If such a dependency exists, the revoke proceeds only if the revokee also holds this privilege from another source (grantor) or holds this privilege indirectly (such as if PUBLIC has this privilege, or if the revokee has SYSADM authority).

For example, consider this scenario.

- UserA creates a user-defined function, UserA.UDFA.
- UserA grants EXECUTE on UserA.UDFA to UserB.
- User B then creates a user-defined function UserB.UDFB that is sourced on UserA.UDFA.

At this point, if UserA attempts to revoke EXECUTE on UserA.UDFA from UserB, the revoke fails with an accompanying message indicating that a dependency exists on this privilege. If, however, UserB had the EXECUTE privilege on UserA.UDFA from another source, directly or indirectly, the EXECUTE privilege that was granted by UserA is revoked successfully.

The objects that are owned by the revokee that can have dependencies on distinct type, JARS (Java classes for a routine), user-defined function, or stored procedure privileges are as follows.

For distinct types:

- A table that has a column that is defined as a distinct type
- A user-defined function that has a parameter that is defined as a distinct type
- A stored procedure that has a parameter that is defined as a distinct type

For user-defined functions:

- Another user-defined function that is sourced on the user-defined function
- A view that uses the user-defined function
- A table that uses the user-defined function in a check constraint or user-defined default clause
- A trigger package that uses the user-defined function

For JAR (Java classes for a routine):

- A Java user-defined function that uses a JAR
- A Java stored procedure that uses a JAR

For stored procedures, a trigger package that refers to the stored procedure in a CALL statement

Another way for the revoke to succeed is to drop the object that has a dependency on the privilege. To determine which objects are dependent on which privileges before attempting the revoke, use the following SELECT statements.

For a distinct type:

- List all tables owned by the revokee USRT002 that contain columns that use the distinct type USRT001.UDT1:

  ```
  SELECT * FROM SYSIBM.SYSCOLUMNS WHERE
         TBCREATOR = 'USRT002'      AND
         TYPESCHEMA = 'USRT001'     AND
         TYPENAME = 'UDT1'          AND
         COLTYPE = 'DISTINCT';
  ```

- List the user-defined functions owned by the revokee USRT002 that contain a parameter defined as distinct type USRT001.UDT1:

  ```
  SELECT * FROM SYSIBM.SYSPARMS WHERE
         OWNER = 'USRT002'          AND
         TYPESCHEMA = 'USRT001'     AND
         TYPENAME = 'UDT1'          AND
         ROUTINETYPE = 'F';
  ```

- List the stored procedures that are owned by the revokee USRT002 that contain a parameter defined as distinct type USRT001.UDT1:

  ```
  SELECT * FROM SYSIBM.SYSPARMS WHERE
         OWNER = 'USRT002'          AND
         TYPESCHEMA = 'USRT001'     AND
         TYPENAME = 'UDT1'          AND
         ROUTINETYPE = 'P';
  ```

For a user-defined function:

- List the user-defined functions that are owned by the revokee USRT002 that are sourced on user-defined function USRT001.SPECUDF1:

  ```
  SELECT * FROM SYSIBM.SYSROUTINES WHERE
         OWNER = 'USRTOO2'            AND
         SOURCESCHEMA = 'USRTOO1'     AND
         SOURCESPECIFIC = 'SPECUDF1'  AND
         ROUTINETYPE = 'F';
  ```

- List the views that are owned by the revokee USRT002 that use user-defined function USRT001.SPECUDF1:

  ```
  SELECT * FROM SYSIBM.SYSVIEWDEP WHERE
         DCREATOR = 'USRTOO2'       AND
         BSCHEMA = 'USRT001'        AND
         BNAME = 'SPECUDF1'         AND
         BTYPE = 'F';
  ```

- List the tables that are owned by the revokee USRT002 that use user-defined function USRT001.A_INTEGER in a check constraint or user-defined default clause:

  ```
  SELECT * FROM SYSIBM.SYSCONSTDEP WHERE
         DTBCREATOR = 'USRT002'      AND
         BSCHEMA = 'USRT001'         AND
         BNAME = 'A_INTEGER'         AND
         BTYPE = 'F';
  ```

- List the trigger packages that are owned by the revokee USRT002 that use user-defined function USRT001.UDF4:

  ```
  SELECT * FROM SYSIBM.SYSPACKDEP WHERE
         DOWNER = 'USRT002'         AND
         BQUALIFIER = 'USRT001'     AND
         BNAME = 'UDF4'             AND
         BTYPE = 'F';
  ```

For a JAR (Java class for a routine):

List the routines owned by the revokee USRT002 that use a JAR named USRT001.SPJAR:

```
SELECT * FROM SYSIBM.SYSROUTINES WHERE
       OWNER = 'USRT002'          AND
       JARCHEMA = 'USRT001'       AND
       JAR_ID = 'SPJAR';
```

For a stored procedure that is used in a trigger package:

List the trigger packages that refer to the stored procedure USRT001.WLMOCN2 that is owned by the revokee USRT002:

```
SELECT * FROM SYSIBM.SYSPACKDEP WHERE
       DOWNER = 'USRT002'         AND
       BQUALIFIER = 'USRT001'     AND
       BNAME = 'WLMLOCN2'         AND
       BTYPE = 'O';
```

## Other implications of the REVOKE statement

***View deletion:*** If a table privilege is revoked from the owner of a view on the table, the corresponding privilege on the view is revoked. The privilege is revoked not only from the owner of the view, but also from all other IDs to which the privilege was granted. If the SELECT privilege on the base table is revoked from the owner of the view, the view is dropped. However, if another grantor granted the SELECT privilege to the view owner before the view was created, the view is not dropped. For example, suppose OPER2 has the SELECT and INSERT privileges on table T1 and creates a view of the table. If the INSERT privilege on T1 is revoked from OPER2, all insert privileges on the view are revoked.If the SELECT privilege on T1 is revoked from OPER2, and if OPER2 did not have the SELECT privilege from another grantor before the view was created, the view is dropped.

If a view uses a user-defined function, the view owner must have the EXECUTE privilege on the function. If the EXECUTE privilege is revoked, the revoke fails, because the view is using the privilege, and the RESTRICT clause prevents the attempt to revoke the privilege.

***Views created by SYSADM:*** An authorization ID with SYSADM authority to create a view for another authorization ID. In this case, the view could have both a creator and an owner. The owner is automatically given the SELECT privilege on the view. However, the privilege on the base table determines whether the view is dropped. For example, suppose that IDADM, with SYSADM authority, creates a view on TABLX with OPER as the owner. OPER now has the SELECT privilege on the view, but not necessarily any privileges on the base table. If SYSADM is revoked from IDADM so that the SELECT privilege on TABLX is gone, the view is dropped.

If one ID creates a view for another, the catalog table SYSIBM.SYSTABAUTH might need two rows to record the fact, as follows:

- If IDADM creates a view for OPER when OPER has enough privileges to create the view by itself, only one row is inserted in SYSTABAUTH. The row shows only that OPER granted the required privileges.
- If IDADM creates a view for OPER when OPER does not have enough privileges to create the view by itself, two rows are inserted in SYSTABAUTH. One row shows IDADM as GRANTOR and OPER as GRANTEE of the SELECT privilege. The other row shows any other privileges that OPER might have on the view because of privileges that are held on the base table.

***Invalidated and inoperative application plans and packages:*** If the owner of an application plan or package loses a privilege that is required by the plan or package, and the owner does not have that privilege from another source, DB2 invalidates the plan or package. For example, suppose OPER2 has the SELECT and INSERT privileges on table T1 and creates a plan that uses SELECT, but not INSERT. If the SELECT privilege is revoked, DB2 invalidates the plan. If the INSERT privilege is revoked, the plan is unaffected. If the revoked privilege was EXECUTE on a user-defined function, DB2 marks the plan or package *inoperative* instead of invalid.

***Implications for caching:*** If authorization data is cached for packages, a revoke of EXECUTE authority on the package from an ID causes that ID to be removed from the cache.

Similarly, if authorization data is cached for routines, a revoke or cascaded revoke of EXECUTE authority on a routine, or on all routines in a schema (schema.*), from any ID causes the ID to be removed from the cache.

If authorization data is cached for plans, a revoke of EXECUTE authority on the plan from any ID causes the authorization cache to be invalidated.

If an application is caching dynamic SQL statements, and a privilege is revoked that was needed when the statement was originally prepared and cached, that statement is removed from the cache. Subsequent PREPARE requests for that statement do not find it in the cache and therefore execute a full PREPARE. If the plan or package is bound with KEEPDYNAMIC(YES), which means the application does not need to explicitly re-prepare the statement after a commit operation, you might get an error on an OPEN, DESCRIBE, or EXECUTE of that statement following the next commit operation. The error can occur because a prepare operation is performed implicitly by DB2. If you no longer have sufficient authority for the prepare, the OPEN, DESCRIBE, or EXECUTE request fails.

***Revoking SYSADM from install SYSADM:*** If you REVOKE SYSADM from the install SYSADM user ID, DB2 does not cascade the revoke. You can therefore change the install SYSADM user ID or delete extraneous SYSADM user IDs. To change the Install SYSADM user ID:

1. Select the new Install SYSADM user ID.
2. GRANT it SYSADM authority.
3. REVOKE SYSADM authority from the old Install SYSADM user ID.
4. Update the SYSTEM ADMIN 1 or 2 field on installation panel DSNTIPP.

To delete an extraneous SYSADM user ID:

1. Write down the current Install SYSADM.
2. Make the SYSADM user ID you want to delete an Install SYSADM ID, by updating the SYSTEM ADMIN 1 or 2 field on installation panel DSNTIPP.
3. REVOKE SYSADM authority from the user ID using another SYSADM user ID.
4. Change the Install SYSADM user ID back to its original value.

## Finding catalog information about privileges

The following catalog tables contain information about the privileges that IDs can hold:

*Table 40. Privileges information in DB2 catalog tables*

| Table name | Records privileges held for |
|---|---|
| SYSIBM.SYSCOLAUTH | Updating columns |
| SYSIBM.SYSDBAUTH | Databases |
| SYSIBM.SYSPLANAUTH | Plans |
| SYSIBM.SYSPACKAUTH | Packages |
| SYSIBM.SYSRESAUTH | Buffer pools, storage groups, collections, table spaces, JARS, and distinct types |
| SYSIBM.SYSROUTINEAUTH | User-defined functions and stored procedures |
| SYSIBM.SYSSCHEMAAUTH | Schemas |
| SYSIBM.SYSTABAUTH | Tables and views |
| SYSIBM.SYSUSERAUTH | System authorities |

For descriptions of the columns of each table, see Appendix D of *DB2 SQL Reference*.

## Retrieving information in the catalog

You can query the DB2 catalog tables by using SQL SELECT statements. Executing those statements requires appropriate privileges and authorities, and you can control access to the catalog by granting and revoking those privileges and authorities. For suggestions about securing the catalog, see “Using views of the DB2 catalog tables” on page 155.

The following examples suggest some of the information you can get from the DB2 catalog.

### Retrieving all DB2 authorization IDs with granted privileges

Some of the catalog tables listed above include columns named GRANTEE and GRANTEETYPE. If GRANTEETYPE is blank, the value of GRANTEE is an ID that has been granted a privilege. No single catalog table contains information about all privileges. However, to retrieve all IDs with privileges, you can issue the following SQL statements:

```
SELECT GRANTEE, 'PACKAGE ' FROM SYSIBM.SYSPACKAUTH
 WHERE GRANTEETYPE = ' ' UNION
SELECT GRANTEE, 'TABLE   ' FROM SYSIBM.SYSTABAUTH
 WHERE GRANTEETYPE = ' ' UNION
SELECT GRANTEE, 'COLUMN  ' FROM SYSIBM.SYSCOLAUTH
 WHERE GRANTEETYPE = ' ' UNION
SELECT GRANTEE, 'ROUTINE ' FROM SYSIBM.SYSROUTINEAUTH
  WHERE GRANTEETYPE = ' ' UNION
SELECT GRANTEE, 'PLAN    ' FROM SYSIBM.SYSPLANAUTH
                         UNION
SELECT GRANTEE, 'SYSTEM  ' FROM SYSIBM.SYSUSERAUTH
                         UNION
SELECT GRANTEE, 'DATABASE' FROM SYSIBM.SYSDBAUTH
                         UNION
SELECT GRANTEE, 'SCHEMA  ' FROM SYSIBM.SYSSCHEMAAUTH
                         UNION
SELECT GRANTEE, 'USE     ' FROM SYSIBM.SYSRESAUTH;
```

Periodically, you should compare the list of IDs that is retrieved by these statements with lists of users from subsystems that connect to DB2—such as IMS, CICS, and TSO—and with lists of RACF groups and lists of users from other DBMSs that access your DB2. If DB2 lists IDs that do not exist elsewhere, you should revoke their privileges.

## Retrieving multiple grants of the same authorization

If several grantors grant the same privilege to the same grantee, the catalog can become cluttered with similar data. This might cause poor performance. (DB2 does not keep duplicate records of the same privilege granted to the same grantee by the same grantor.) However, you might want authority granted from several different IDs. For example, you might want an ID to retain a privilege that is revoked by just one of the sources that granted it.

The following SQL statement retrieves duplicate grants on plans. If multiple grants clutter your catalog, examine the output from a query like this one, starting at the top with the most frequent grants.

```
SELECT GRANTEE, NAME, COUNT(*)
  FROM SYSIBM.SYSPLANAUTH
  GROUP BY GRANTEE, NAME
  HAVING COUNT(*) > 2
  ORDER BY 3 DESC;
```

Similar statements for other catalog tables can retrieve information about multiple grants on other types of objects.

## Retrieving all IDs with DBADM authority

To retrieve all IDs that have DBADM authority, issue:

```
SELECT DISTINCT GRANTEE
  FROM SYSIBM.SYSDBAUTH
  WHERE DBADMAUTH <>' ' AND GRANTEETYPE = ' ';
```

## Retrieving IDs authorized to access a table

To retrieve all IDs that are explicitly authorized to access the employee table (DSN8710.EMP in database DSN8D71A), issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
  WHERE TTNAME = 'EMP' AND TCREATOR = 'DSN8710'
  AND GRANTEETYPE = ' ';
```

To find out who can change the employee table, issue the following statement. It retrieves IDs with administrative authorities, as well as IDs to which authority is explicitly granted.

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
  WHERE TTNAME = 'EMP' AND TCREATOR = 'DSN8710' AND
    GRANTEETYPE = ' ' AND
       (ALTERAUTH  <> ' ' OR
        DELETEAUTH <> ' ' OR
        INSERTAUTH <> ' ' OR
        UPDATEAUTH <> ' ')
UNION
SELECT GRANTEE FROM SYSIBM.SYSUSERAUTH
  WHERE SYSADMAUTH <> ' '
UNION
SELECT GRANTEE FROM SYSIBM.SYSDBAUTH
  WHERE DBADMAUTH <> ' ' AND NAME = 'DSN8D71A';
```

To retrieve the columns of DSN8710.EMP for which update privileges have been granted on a specific set of columns, issue the following statement:

```
SELECT DISTINCT COLNAME, GRANTEE, GRANTEETYPE FROM SYSIBM.SYSCOLAUTH
  WHERE CREATOR='DSN8710' AND TNAME='EMP'
  ORDER BY COLNAME;
```

The character in the GRANTEETYPE column shows whether the privileges have been granted to an authorization ID (blank) or are used by an application plan or package (P).

To retrieve the IDs that have been granted the privilege of updating one or more columns of DSN8710.EMP, issue the following statement:

```
SELECT DISTINCT GRANTEE
   FROM SYSIBM.SYSTABAUTH
     WHERE TTNAME = 'EMP' AND TCREATOR='DSN8710' AND GRANTEETYPE=' '
     AND UPDATEAUTH <> ' ';
```

The query returns only the IDs to which update privileges have been specifically granted. It does not return those who have the privilege because of SYSADM or DBADM authority. You could include them by forming the union with another query.

## Retrieving IDs authorized to access a routine

To retrieve the IDs that are authorized to access stored procedure PROCA in schema SCHEMA1, issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSROUTINEAUTH
  WHERE SPECIFICNAME='PROCA' AND SCHEMA='SCHEMA1' AND GRANTEETYPE=' '
  AND ROUTINETYPE ='P';
```

You can write a similar statement to retrieve the IDs that are authorized to access a user-defined function. In this case, the value for ROUTINETYPE is 'F'.

## Retrieving the tables an ID is authorized to access

To retrieve the list of tables and views that PGMR001 can access, issue the following statement:

```
SELECT DISTINCT TCREATOR, TTNAME FROM SYSIBM.SYSTABAUTH
  WHERE GRANTEE = 'PGMR001' AND GRANTEETYPE =' ';
```

To retrieve the tables, views, and aliases that PGMR001 owns, issue the following statement:

```
SELECT NAME FROM SYSIBM.SYSTABLES
  WHERE CREATOR = 'PGMR001';
```

## Retrieving the plans and packages that access a table

To retrieve the names of application plans and packages that refer to table DSN8710.EMP directly, issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
  WHERE GRANTEETYPE = 'P'       AND
        TCREATOR    = 'DSN8710'  AND
        TTNAME      = 'EMP';
```

A plan or package can refer to the table indirectly, through a view. To find all views that refer to the table, query SYSIBM.SYSVIEWDEP. Then find all plans and packages that refer to those views by issuing statements like the one above.

The query above does not distinguish between plans and packages. To identify a package, use the COLLECTION column of table SYSTABAUTH, which names the collection a package resides in and is blank for a plan.

## Using views of the DB2 catalog tables

Only an ID with SYSADM or SYSCTRL authority automatically has the privilege of retrieving data from catalog tables. If you do not want to grant the SELECT privilege on all catalog tables to PUBLIC, consider using views to let each ID retrieve information about its own privileges.

For example, the following view includes the owner and the name of every table on which a user's primary authorization ID has the SELECT privilege:

```
CREATE VIEW MYSELECTS AS
  SELECT TCREATOR, TTNAME FROM SYSIBM.SYSTABAUTH
    WHERE SELECTAUTH <> ' ' AND GRANTEETYPE = ' ' AND
      GRANTEE IN (USER, 'PUBLIC', 'PUBLIC*', CURRENT SQLID);
```

The keyword USER in that statement is equal to the value of the primary authorization ID. To include tables that can be read by a secondary ID, set the current SQLID to that secondary ID before querying the view.

To make the view available to every ID, issue:

```
GRANT SELECT ON MYSELECTS TO PUBLIC;
```

Similar views can show other privileges. This one shows privileges over columns:

```
CREATE VIEW MYCOLS (OWNER, TNAME, CNAME, REMARKS, LABEL)
 AS SELECT DISTINCT TBCREATOR, TBNAME, NAME, REMARKS, LABEL
 FROM SYSIBM.SYSCOLUMNS, SYSIBM.SYSTABAUTH
  WHERE TCREATOR = TBCREATOR AND TTNAME = TBNAME AND GRANTEETYPE = '   '
    AND GRANTEE IN (USER,'PUBLIC',CURRENT SQLID,'PUBLIC*');
```

# Chapter 11. Controlling access through a closed application

A *closed application* is an application that requires DB2 objects to be managed solely through external interfaces. As an example, consider an application process that uses DB2 as a repository for changing data. The process does not merely write to and read from a fixed set of tables; it must also create, alter, and drop tables, and perhaps other objects, to deal with new data formats. Normally, a database administrator would have the privileges needed to do those operations at any time, but now the operations must be done only through a specific application. The application is “closed” because it requires exclusive control over data definition statements for some set of objects.

If you install *data definition control support* on installation panel DSNTIPZ, you can control how specific plans or package collections can use those statements. Figure 14 lists the specific statements that are controlled. In this chapter, those statements are referred to as “data definition language”, or “DDL”.

The control does not avoid existing authorization checks; it does impose additional constraints. You register plans and package collections in a special table, and you register the objects that the plans and collections that are associated with another table. DB2 then consults those two registration tables before accepting a given DDL statement from a process. If the registration tables indicate that the particular process is not allowed to create, alter, or drop that particular object, DB2 does not allow it.

This chapter tells how to impose several degrees of control over applications and objects; see “Controlling data definition”.

If you choose to impose those controls, you have two tables to manage: the *application registration table* (ART) and the *object registration table* (ORT). For instructions, see “Managing the registration tables and their indexes” on page 164.

```
CREATE ALIAS                             DROP ALIAS         COMMENT ON
CREATE DATABASE     ALTER DATABASE       DROP DATABASE      LABEL ON
CREATE INDEX        ALTER INDEX          DROP INDEX
CREATE STOGROUP     ALTER STOGROUP       DROP STOGROUP
CREATE SYNONYM                           DROP SYNONYM
CREATE TABLE        ALTER TABLE          DROP TABLE
CREATE TABLESPACE   ALTER TABLESPACE     DROP TABLESPACE
CREATE VIEW                              DROP VIEW
```

*Figure 14. Statements controlled by data definition control support*

## Controlling data definition

You can control the use of data definition language through several installation options and entries in two special tables, the ART and the ORT. In those tables, you register the names of plans and package collections that make up an application and the names of the objects whose data definition they control. First, you choose installation options to make any use of data definition control; see “Required installation options” on page 158. The next sections illustrate the use of installation options and the registration tables for the following situations:

- Control by application name
  - – Registered applications have total control over all DDL in the DB2 subsystem. See “Controlling by application name” on page 158.

  - Registered applications have total control with some exceptions. See “Controlling by application name with exceptions” on page 160.
- Control by object name
  - All objects in the system are registered and controlled by name. See “Controlling by object name” on page 162.
  - Some specific objects are registered and controlled. DDL is accepted for objects that are not registered. See “Controlling by object name with exceptions” on page 163.

The names in some columns in the ART and ORT can be represented by patterns that use the percent sign (%) and the underscore (_) characters. “Using name patterns” on page 159 tells you how to do this.

## Required installation options

To use the ART and ORT, you must install data definition control support by entering YES for the first option as follows:

```
1 INSTALL DD CONTROL SUPT. ===> YES
```

Also on panel DSNTIPZ, choose the names for the registration tables in your DB2 subsystem, their owners, and the databases they reside in. You can accept the default names or assign names of your own. The default names are as follows:

```
6 REGISTRATION OWNER       ===> DSNRGCOL
7 REGISTRATION DATABASE    ===> DSNRGFDB
8 APPL REGISTRATION TABLE  ===> DSN_REGISTER_APPL
9 OBJT REGISTRATION TABLE  ===> DSN_REGISTER_OBJT
```

This chapter uses these default names. If you specify different table names, each name can have a maximum of 17 characters.

Four other options on installation panel DSNTIPZ, which are described later in this chapter, determine how DDL statements are controlled:

```
 2 CONTROL ALL APPLICATIONS ===>
 3 REQUIRE FULL NAMES       ===>
 4 UNREGISTERED DDL DEFAULT ===>
 5 ART/ORT ESCAPE CHARACTER ===>
```

## Controlling by application name

The simplest use of data definition control is to give one or more applications total control over the use of DDL in the system. To do that:

1. When installing DB2, choose to control all applications. On panel DSNTIPZ, specify:

   ```
   CONTROL ALL APPLICATIONS ===> YES
   ```

   That choice allows only package collections or plans that are registered in the ART to use DDL statements. (This case, then, does not require any use of the ORT.)
2. Register, in the ART, all package collections that you allow to issue DDL statements, using the value Y in column DEFAULTAPPL. If a plan is to issue DDL statements that are not bound to a package, register the plan name. You must supply values for at least the following columns:

   **Column name** **Description**

   **APPLIDENT** Collection-ID of the package that is executing the DDL or, if no package exists, the name of the plan that is executing the DDL

**APPLIDENTTYPE**
Type of item named by APPLIDENT:

**P** Application plan

**C** Package collection

**DEFAULTAPPL**
Indicates whether the plan or package collection named by APPLIDENT can use DDL. Enter Y (Yes); the default is N (No).

(You can enter information in other columns for your own use. For a complete description of the table, see “Columns of the ART” on page 164.)

***Example:*** Suppose you want all DDL in your system to be issued only through certain applications. The applications are identified by:
1. PLANA, the name of an application plan
2. PACKB, a package collection-ID
3. TRULY%, a pattern for any plan name beginning with TRULY
4. TR%, a pattern for any plan name beginning with TR

Table 41 shows the entries you need in your ART.

*Table 41. Table DSN_REGISTER_APPL for total system control*

| APPLIDENT | APPLIDENTTYPE | DEFAULTAPPL |
|---|---|---|
| PLANA | P | Y |
| PACKB | C | Y |
| TRULY% | P | Y |
| TR% | P | N |

***Using name patterns:*** DB2 accepts two pattern characters:
- The percent sign (%), to represent zero or more characters
- The underscore character (_), to represent a single character

Patterns are used here much as they are in the SQL LIKE predicate described in Chapter 2 of *DB2 SQL Reference*. However, the one difference is that blanks following a pattern character are not significant. DB2 treats 'A% ' the same as 'A%'.

***The escape character:*** If you want the percent or underscore character to be treated as a character, specify an escape character for option 5 on installation panel DSNTIPZ. The escape character can be any special character, except underscore (_) or percent (%). For example, to use the pound sign (#), specify:

```
5 ART/ORT ESCAPE CHARACTER ===> #
```

With that specification, the pound sign can be used in names in the same way as an escape character is used in an SQL LIKE predicate.

***An inactive table entry:*** If the row with TR% for APPLIDENT in Table 41 originally contains the value Y for DEFAULTAPPL, any plan with a name beginning with TR can execute DDL. Then if DEFAULTAPPL is changed to N to disallow that use, the changed row *does not prevent* plans beginning with TR from using DDL; the row merely fails to allow that use. (When the table is checked, that row is ignored.) Hence, the plan TRULYXYZ is allowed to use DDL, by the row with APPLIDENT TRULY%.

## Controlling by application name with exceptions

In this situation, you want to give one or more applications *almost* total control over DDL. You reserve only a few objects that are to be created, altered, or dropped by other applications. To do that:

1. When installing DB2, choose *not* to control all applications. On panel DSNTIPZ, specify:

   ```
   CONTROL ALL APPLICATIONS ===> NO
   ```

   That choice allows unregistered applications to use DDL statements. The ORT determines restrictions that apply to that use.

2. Also on panel DSNTIPZ, specify:

   ```
   UNREGISTERED DDL DEFAULT ===> APPL
   ```

   That choice restricts the use of DDL statements for objects that are *not* registered in the ORT: only registered applications can use DDL for unregistered objects. Hence, the registered applications retain almost total control; only registered objects are possible exceptions.

3. In the ORT, register all objects that are exceptions to the system DDL control. You must supply values for at least the following columns:

| Column name | Description |
|---|---|
| **QUALIFIER** | Qualifier for the object name |
| **NAME** | Simple name of the object |
| **TYPE** | Type of named object:<br>**C** Table, view, index, synonym, or alias<br>**D** Database<br>**T** Table space<br>**S** Storage group |
| **APPLMATCHREQ** | Indicates whether only the application named in APPLIDENT can use DDL for this object: Y (Yes) or N (No) |
| **APPLIDENT** | Collection-ID of the package that can have exclusive control over DDL for this object or, if no package exists, the name of the plan that can have exclusive control |
| **APPLIDENTTYPE** | Type of item named by APPLIDENT:<br>**P** Application plan<br>**C** Package collection |

   (You can enter information in other columns for your own use. For a complete description of the table, see “Columns of the ORT” on page 165.)

***Example:*** Suppose that you want *almost* all DDL in your system to be issued only through certain applications, known by an application plan (PLANA), a package collection (PACKB), and a pattern for plan names (TRULY%). However, you also want these specific exceptions:

The ART remains as in Table 41 on page 159; PLANA and PACKB have total system control (but with exceptions). Table 42 on page 161 shows the entries that are needed to register those exceptions in the ORT.

*Table 42. Table DSN_REGISTER_OBJT for system control with exceptions*

| QUALIFIER | NAME | TYPE | APPLMATCHREQ | APPLIDENT | APPLIDENTTYPE |
|---|---|---|---|---|---|
| KIM [1] | VIEW1 | C | Y | PLANC | P |
| BOB [2] | ALIAS | C | Y | PACKD | C |
| FENG [3] | TABLE2 | C | N | | |
| SPIFFY [4] | MSTR_ | C | Y | TRULY% | P |

**Notes:**

1. Requires an application match for the object KIM.VIEW1: the view can be created, altered, or dropped only by the application plan PLANC.
2. Specifies that BOB.ALIAS can be created, altered, or dropped only by the package collection PACKD.
3. Requires no application match for FENG.TABLE2: the object can be created, altered, or dropped by *any* plan or package collection.
4. The fourth entry requires only a pattern match; the object SPIFFY.MSTRA, for example, can be created, altered, or dropped by plan TRULYJKL.

## Registering sets of objects

Complete two-part names are not required for every object that is registered in the ORT. To use incomplete names, on installation panel DSNTIPZ specify:

```
3 REQUIRE FULL NAMES       ===> NO
```

The default value, YES, requires you to use both parts of the name of each registered object. With the value NO, an incomplete name in the ORT represents a set of objects that all share the same value for one part of a two-part name. Objects that are represented by incomplete names in the ORT need an authorizing entry in the ART.

The entries shown in Table 43 can be added to Table 42 when NO is specified:

*Table 43. Table DSN_REGISTER_OBJT for objects with incomplete names*

| QUALIFIER | NAME | TYPE | APPLMATCHREQ | APPLIDENT | APPLIDENTTYPE |
|---|---|---|---|---|---|
| | TABA | C | Y | PLANX | P |
| | TABB | C | Y | PACKY | C |
| SYSADM | | C | N | | |
| DBSYSADM | | T | N | | |
| USER1 | TABLEX | C | N | | |

The first two entries record two sets of objects, *.TABA and *.TABB, which are controlled by PLANX and PACKY, respectively. That is, only PLANX can create, alter, or drop any object whose name is *qual*.TABA, where *qual* is any appropriate qualifier. Only PACKY can create, alter, or drop any object whose name is *qual*.TABB. PLANX and PACKY must also be registered in the ART with QUALIFIEROK set to Y, as shown in Table 44 on page 162. That allows the applications to use sets of objects that are registered in the ORT with an incomplete name.

The next two new entries in the ORT record:
1. Tables, views, indexes, or aliases with names like SYSADM.*
2. Table spaces with names like DBSYSADM.*; that is, table spaces in database DBSYSADM

The last entry in the ORT allows two kinds of incomplete names: table names like USER1.* and table names like *.TABLEX.

***ART entries for objects with incomplete names in the ORT:*** Objects having names like those patterns can be created, altered, or dropped by any package collection or application plan, because APPLMATCHREQ = N. However, the collection or plan that creates, alters, or drops such an object must be registered in the ART with QUALIFIEROK=Y, to allow it to use incomplete object names.

Table 44 shows PLANA and PACKB registered in the ART to use sets of objects that are registered in the ORT with incomplete names.

*Table 44. Table DSN_REGISTER_APPL for plans that use sets of objects*

| APPLIDENT | APPLIDENTTYPE | DEFAULTAPPL | QUALIFIEROK |
|---|---|---|---|
| PLANA | P | N | Y |
| PACKB | C | N | Y |

## Controlling by object name

In this situation, you want each of several applications to control a specific set of objects, and you want no unregistered objects in the system. You do allow some registered objects that are not controlled by specific applications. To accomplish that:

1. When installing DB2, choose not to control all applications, as in “Controlling by application name with exceptions” on page 160. On panel DSNTIPZ, specify:

   ```
   CONTROL ALL APPLICATIONS ===> NO
   ```

2. Also on panel DSNTIPZ, specify:

   ```
   UNREGISTERED DDL DEFAULT ===> REJECT
   ```

   That option totally restricts the use of DDL statements for objects that are not registered in the ORT: *no* application can create, or use any DDL, for any unregistered object. (This case, then, might not require any use of the ART.)
3. Register all objects in the system in the ORT by QUALIFIER, NAME, and TYPE. You can use name patterns for QUALIFIER and NAME. (If you used REQUIRE FULL NAMES = NO, register sets of objects by NAME and TYPE or by QUALIFIER and TYPE.) For each controlled object, use APPLMATCHREQ = Y. Give the name of the plan or package collection that controls the object in the APPLIDENT column. (Again, you can use a name pattern.) You can have only one row in the ORT for each combination of QUALIFIER.NAME.TYPE.
4. Register in the ART, with QUALIFIEROK = Y, any plan or package collection that can use a set of objects that you register in the ORT with an incomplete name, regardless of whether that set has APPLMATCHREQ = Y.

***Example:*** Table 45 on page 163 shows entries in the ORT for a DB2 subsystem containing the following objects:

- Two storage groups and a database that are not controlled by a specific application. Those could be created, altered, or dropped by a user with the appropriate authority using any application, such as SPUFI or QMF.
- Two table spaces that are not controlled by a specific application. Their names are qualified by the name of the database they reside in.
- Three objects whose names are qualified by the authorization IDs of their owners. Those objects could be tables, views, indexes, synonyms, or aliases. DDL statements for those objects can be issued only through the application plan named PLANX or the package collection named PACKX.

- Objects with names like EDWARD.OBJ4, ED.OBJ4, and EBHARD.OBJ4, that can be created, altered, or deleted by application plan SPUFI. Entry E%D in the QUALIFIER column represents all three objects.
- Objects with names beginning TRULY.MY_, where the underscore character is actually part of the name. Assuming that you specified # as the escape character, all of those objects can be created, altered, or dropped only by plans with names that begin with TRULY.

Assume the following installation option:

```
REQUIRE FULL NAMES       ===> YES
```

Entries in Table 45 do not specify incomplete names. Hence, objects that are not represented in the table cannot be created in the system, except by an ID with installation SYSADM authority.

*Table 45. Table DSN_REGISTER_OBJT for total control by object*

| QUALIFIER | NAME | TYPE | APPLMATCHREQ | APPLIDENT | APPLIDENTTYPE |
|---|---|---|---|---|---|
| | STOG1 | S | N | | |
| | STOG2 | S | N | | |
| | DATB1 | D | N | | |
| DATB1 | TBSP1 | T | N | | |
| DATB1 | TBSP2 | T | N | | |
| KIM | OBJ1 | C | Y | PLANX | P |
| FENG | OBJ2 | C | Y | PLANX | P |
| QUENTIN | OBJ3 | C | Y | PACKX | C |
| E%D | OBJ4 | C | Y | SPUFI | P |
| TRULY | MY#_% | C | Y | TRULY% | P |

## Controlling by object name with exceptions

In this situation, you want each of several applications to control a specific set of registered objects. You also allow other applications to use DDL statements for unregistered objects.

1. When installing DB2, choose not to control all applications, as in “Controlling by application name with exceptions” on page 160. On panel DSNTIPZ, specify:

   ```
   CONTROL ALL APPLICATIONS ===> NO
   ```

2. Also on panel DSNTIPZ, specify:

   ```
   UNREGISTERED DDL DEFAULT ===> ACCEPT
   ```

   That option *does not* restrict the use of DDL statements for objects that are not registered in the ORT: *any* application can use DDL for any unregistered object.

3. Register all controlled objects in the ORT. Use a name and qualifier to identify a single object. Use only one part of a two-part name to identify a set of objects that share just that part of the name. For each controlled object, use APPLMATCHREQ = Y. Give the name of the plan or package collection that controls the object in the APPLIDENT column.
4. For each set of controlled objects (identified by only a simple name in the ORT), register the controlling application in the ART. Supply values for the APPLIDENT and APPLIDENTTYPE columns as in Table 44 on page 162. You must also supply values for one additional column:

   **Column name**

   **Description**

**QUALIFIEROK**

Specify Y (Yes) to show that the application can supply the remaining part of the name in DDL statements for objects that are registered in the ORT by an incomplete name.

***Example:*** The two tables below assume that the installation option, REQUIRE FULL NAMES, is set to NO, as described in “Registering sets of objects” on page 161. Table 46 shows entries in the ORT for the following controlled objects:

- The objects KIM.OBJ1, FENG.OBJ2, QUENTIN.OBJ3, and EDWARD.OBJ4, all of which are controlled by PLANX or PACKX, as described under “Controlling by object name” on page 162. DB2 cannot interpret the object names as incomplete names, because the objects that control them, PLANX and PACKX, are registered in Table 47 with QUALIFIEROK=N.
- Two sets of objects, *.TABA and *.TABB, which are controlled by PLANA and PACKB, respectively.

*Table 46. Table DSN_REGISTER_OBJT for object control with exceptions*

| QUALIFIER | NAME | TYPE | APPLMATCHREQ | APPLIDENT | APPLIDENTTYPE |
|---|---|---|---|---|---|
| KIM | OBJ1 | C | Y | PLANX | P |
| FENG | OBJ2 | C | Y | PLANX | P |
| QUENTIN | OBJ3 | C | Y | PACKX | C |
| EDWARD | OBJ4 | C | Y | PACKX | C |
|  | TABA | C | Y | PLANA | P |
|  | TABB | C | Y | PACKB | C |

Table 47 shows entries in the corresponding ART:

*Table 47. Table DSN_REGISTER_APPL for object control with exceptions*

| APPLIDENT | APPLIDENTTYPE | DEFAULTAPPL | QUALIFIEROK |
|---|---|---|---|
| PLANX | P | N | N |
| PACKX | C | N | N |
| PLANA | P | N | Y |
| PACKB | C | N | Y |

In this situation, with the combination of installation options shown above, any application can use DDL for objects that are *not* covered by entries in the ORT. For example, if user HOWARD has the CREATETAB privilege, he can create the table HOWARD.TABLE10 through any application.

# Managing the registration tables and their indexes

“Columns of the ART” and “Columns of the ORT” on page 165 describe the columns of the two registration tables.

## An overview of the registration tables

### Columns of the ART

*Table 48. Columns of the ART*

| Column | Column name | Description |
|---|---|---|
| 1 | APPLIDENT | Collection-ID of the package executing the DDL or, if no package exists, the name of the plan that executes the DDL |

*Table 48. Columns of the ART (continued)*

| | | |
|---|---|---|
| 2 | APPLIDENTTYPE | Type of application identifier |
| 3 | APPLICATIONDESC | Optional data. See “Columns for optional use” on page 167. |
| 4 | DEFAULTAPPL | Indicates whether all DDL should be accepted from this application |
| 5 | QUALIFIEROK | Indicates whether the application can supply a missing name part for objects that are named in the ORT, if REQUIRE FULL NAMES = NO |
| 6 | CREATOR | Optional data. See “Columns for optional use” on page 167. |
| 7 | CREATETIMESTAMP | Optional data. See “Columns for optional use” on page 167. |
| 8 | CHANGER | Optional data. See “Columns for optional use” on page 167. |
| 9 | CHANGETIMESTAMP | Optional data. See “Columns for optional use” on page 167. |

## Columns of the ORT

*Table 49. Columns of the ORT*

| Column | Column name | Description |
|---|---|---|
| 1 | QUALIFIER | Object name qualifier |
| 2 | NAME | Unqualified object name |
| 3 | TYPE | Type of object |
| 4 | APPLMATCHREQ | Indicates whether an application that names this object must match the one named in the APPLIDENT column |
| 5 | APPLIDENT | Collection-ID of the plan or package that executes the DDL |
| 6 | APPLIDENTTYPE | Type of application identifier |
| 7 | APPLICATIONDESC | Optional data. See “Columns for optional use” on page 167. |
| 8 | CREATOR | Optional data. See “Columns for optional use” on page 167. |
| 9 | CREATETIMESTAMP | Optional data. See “Columns for optional use” on page 167. |
| 10 | CHANGER | Optional data. See “Columns for optional use” on page 167. |
| 11 | CHANGETIMESTAMP | Optional data. See “Columns for optional use” on page 167. |

## Creating the tables and indexes

The ART, the ORT, and the required unique indexes on them are created when you install data definition control support. If you drop any of those objects, you can re-create them using the CREATE statements shown here:

**CREATE statements for the ART and its index:**

```
CREATE TABLE DSNRGCOL.DSN_REGISTER_APPL
  (APPLIDENT         CHAR(18)     NOT NULL WITH DEFAULT,
   APPLIDENTTYPE     CHAR(1)      NOT NULL WITH DEFAULT,
   APPLICATIONDESC   VARCHAR(30)  NOT NULL WITH DEFAULT,
   DEFAULTAPPL       CHAR(1)      NOT NULL WITH DEFAULT,
   QUALIFIEROK       CHAR(1)      NOT NULL WITH DEFAULT,
   CREATOR           CHAR(26)     NOT NULL WITH DEFAULT,
   CREATETIMESTAMP   TIMESTAMP    NOT NULL WITH DEFAULT,
   CHANGER           CHAR(26)     NOT NULL WITH DEFAULT,
   CHANGETIMESTAMP   TIMESTAMP    NOT NULL WITH DEFAULT)
  IN DSNRGFDB.DSNRGFTS;
```

```
CREATE UNIQUE INDEX DSNRGCOL.DSN_REGISTER_APPLI
  ON DSNRGCOL.DSN_REGISTER_APPL
  (APPLIDENT, APPLIDENTTYPE, DEFAULTAPPL DESC, QUALIFIEROK DESC)
  CLUSTER;
```

**CREATE statements for the ORT and its index:**

```
CREATE TABLE DSNRGCOL.DSN_REGISTER_OBJT
  (QUALIFIER         CHAR(8)      NOT NULL WITH DEFAULT,
   NAME              CHAR(18)     NOT NULL WITH DEFAULT,
   TYPE              CHAR(1)      NOT NULL WITH DEFAULT,
   APPLMATCHREQ      CHAR(1)      NOT NULL WITH DEFAULT,
   APPLIDENT         CHAR(18)     NOT NULL WITH DEFAULT,
   APPLIDENTTYPE     CHAR(1)      NOT NULL WITH DEFAULT,
   APPLICATIONDESC   VARCHAR(30)  NOT NULL WITH DEFAULT,
   CREATOR           CHAR(26)     NOT NULL WITH DEFAULT,
   CREATETIMESTAMP   TIMESTAMP    NOT NULL WITH DEFAULT,
   CHANGER           CHAR(26)     NOT NULL WITH DEFAULT,
   CHANGETIMESTAMP   TIMESTAMP    NOT NULL WITH DEFAULT)
  IN DSNRGFDB.DSNRGFTS;
```

```
CREATE UNIQUE INDEX DSNRGCOL.DSN_REGISTER_OBJTI
  ON DSNRGCOL.DSN_REGISTER_OBJT
  (QUALIFIER, NAME, TYPE) CLUSTER;
```

You can alter these statements to add columns to the ends of the tables, assign an auditing status, or choose buffer pool or storage options for indexes. You can create these tables with table check constraints to limit the types of entries that are allowed. If you change either of the table names, their owner, or their database, you must reinstall DB2 in update mode and make the corresponding changes on panel DSNTIPZ. Name the required index by adding the letter I to the corresponding table.

Every member of a data sharing group must have the same names for the ART and ORT tables

If you drop any of the registration tables or indexes, most data definition statements are rejected until the dropped objects are re-created. The only DDL statements that are allowed in such circumstances are those that create the registration tables that are defined during installation, their indexes, and the table spaces and database that contain them.

The installation job DSNTIJSG creates a segmented table space to hold the ART and the ORT, using this statement:

```
CREATE TABLESPACE DSNRGFTS IN DSNRGFDB SEGSIZE 4 CLOSE NO;
```

If you want to use a table space with a different name or different attributes, you can modify job DSNTIJSG before installing DB2 or else drop the table space and re-create it, the two tables, and their indexes.

## Adding columns

You can add columns to either registration table for your own use, using the ALTER TABLE statement. If IBM adds columns to either table in future releases, the column names will contain only letters and numbers; consider using some special character, such as the plus sign (+), in your column names to avoid possible conflict.

## Updating the tables

You can load either table with the LOAD utility or update it with SQL INSERT, UPDATE, or DELETE statements. Security provisions are important. Allow only a restricted set of authorization IDs, or perhaps only those with SYSADM authority, to update the ART. Consider assigning a validation exit routine to the ORT, to allow applications to change only those rows that have the same application identifier in the APPLIDENT column. A registration table cannot be updated until all jobs whose DDL statements are controlled by the table have completed.

## Columns for optional use

The ART and ORT contain columns that are not used by DB2. **Recommendation:** Use these columns to audit and manage the tables as follows:

- In APPLICATIONDESC, put a more readable description of each application than the eight-character APPLIDENT column can contain.
- In CREATOR or CHANGER, put the authorization ID that created or last changed the row. The columns are large enough for a three-part name, with the parts separated by periods in columns 9 and 18. If you enter only the primary authorization ID (from the SQL value USER), consider entering it right-justified in the field—that is, preceded by 18 blanks.
- When updating CREATETIMESTAMP and CHANGETIMESTAMP, enter CURRENT TIMESTAMP. When you load or insert a row, DB2 can automatically enter the value of CURRENT TIMESTAMP.

## Stopping data definition control

When data definition control is active, only the users with installation SYSADM or installation SYSOPR authority are able to stop the database, a table space, or an index space containing a registration table or index. When the object is stopped, only an ID with one of those authorities can start it again.

***Bypassing data definition control:*** An ID with install SYSADM authority can execute DDL statements regardless of whether data definition control is active, and of whether the ART or ORT is available, through the following means:

- Through a static SQL statement, if the ID is owner of the plan or package that contains the statement
- Through a dynamic CREATE statement, if the ID is the current SQLID
- Through a dynamic ALTER or DROP statement, if the ID is the current SQLID, the primary ID, or any secondary ID of the executing process

# Chapter 12. Controlling access to a DB2 subsystem

This chapter tells how to control access to the DB2 subsystem from different environments and how to associate a process with an intended set of authorization IDs.

***Recommendation for external security system:*** Control access through an external security system, for which Resource Access Control Facility (RACF) is the model. "Establishing RACF protection for DB2" on page 198 tells how to make DB2 and its IDs known to RACF.

Control by RACF is not strictly necessary, and some alternatives are described under "Other methods of controlling access" on page 214. However, most of the description assumes that RACF, or an equivalent product, is already in place.

***Local requests only:*** If you are not accepting requests from or sending requests to remote locations, begin this chapter with "Controlling local requests". When you come to "Controlling requests from remote applications" on page 176, you can skip everything up to "Establishing RACF protection for DB2" on page 198.

***Remote requests:*** If you are accepting requests from remote applications, you might first want to read "Controlling requests from remote applications" on page 176, which describes the security checks that a remote request is subject to before it can access your DB2 subsystem. The level of security differs depending on whether the requesting application is using SNA or Transmission Control Protocol/Internet Protocol (TCP/IP) protocols to access DB2. After the incoming ID has been authenticated by the local system, the ID is treated like a local connection request or a local sign-on request: You can process it with your connection or sign-on exit routine and associate secondary authorization IDs with it. For more information, see "Controlling local requests".

If you are sending requests to a remote DB2 subsystem, that subsystem can subject your requests to various security checks. For suggestions on how to plan for those checks, see "Planning to send remote requests" on page 189. If you send requests to a remote DBMS that is not DB2 for OS/390 and z/OS, use the documentation for that DRDA database server.

***Topics covered in this chapter:***

## Controlling local requests

Different local processes enter the access control procedure at different points, depending on the environment where they originate. (Quite different criteria apply to remote requests; they are described in "Controlling requests from remote applications" on page 176.)

- The following processes go through connection processing only:

- Requests originating in TSO foreground and background (including online utilities and requests through the call attachment facility)
- JES-initiated batch jobs
- Requests through started task control address spaces (from the MVS START command)

- The following processes go through connection processing and can later go through the sign-on exit also.
  - The IMS control region.
  - The CICS recovery coordination task.
  - DL/I batch.
  - Applications that connect using the Recoverable Resource Manager Services attachment facility (RRSAF). (See Part 6 of *DB2 Application Programming and SQL Guide* for more information.)
- The following processes go through sign-on processing:
  - Requests from IMS dependent regions (including MPP, BMP, and Fast Path)
  - CICS transaction subtasks

For instructions on controlling the IDs that are associated with connection requests, see “Processing connections”. For instructions on controlling the IDs that are associated with sign-on requests, see “Processing sign-ons” on page 173.

IMS, CICS, RRSAF, or DDF-to-DDF connections can send a sign-on request, typically in order to execute an application plan. That request must provide a primary ID; optionally, it can provide secondary IDs also. After a plan is allocated, it need not be deallocated until a new plan is needed. A different transaction can use the same plan by issuing a new sign-on request with a new primary ID.

## Processing connections

A connection request makes a new connection to DB2; it does not reuse an application plan that is already allocated. Therefore, an essential step in processing the request is to check that the ID is authorized to use DB2 resources, as shown in Figure 15.

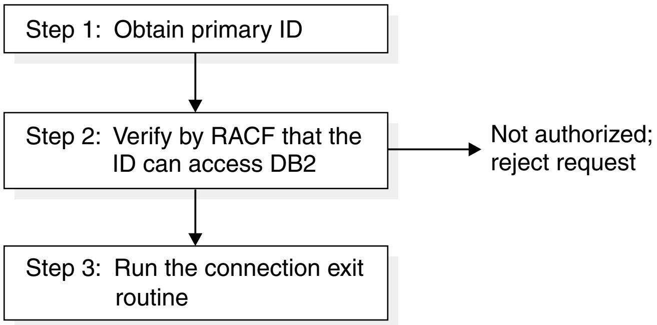

Step 1: Obtain primary ID
Step 2: Verify by RACF that the ID can access DB2
Not authorized; reject request
Step 3: Run the connection exit routine

*Figure 15. Connection processing*

## The steps in detail

The steps in processing connections are:

1. DB2 obtains the initial primary ID. Table 50 on page 171 shows how the source of the ID depends on the type of address space from which the connection was made.

*Table 50. Sources of initial primary authorization identifiers*

| Source | Initial primary authorization ID |
|---|---|
| TSO | TSO logon ID. |
| BATCH | USER parameter on JOB statement. |
| IMS control region or CICS | USER parameter on JOB statement. |
| IMS or CICS started task | Entries in the started task control table. |
| Remote access requests | Depends on the security mechanism used. See “Overview of security mechanisms for DRDA and SNA” on page 176 for more details. |

2. RACF is called through the MVS system authorization facility (SAF) to check whether the ID that is associated with the address space is authorized to use:

   The DB2 resource class (CLASS=DSNR)
   The DB2 subsystem (SUBSYS=*ssnm*)
   The connection type requested

   For instructions on authorizing those uses, see “Permitting RACF access” on page 202. The SAF return code (RC) from the invocation determines the next step, as follows:

   **If RC > 4**, RACF determined that the RACF user ID is not valid or does not have the necessary authorization to access the resource name; DB2 rejects the request for a connection.

   **If RC = 4**, the RACF return code is checked. If that value is:

   **= 4**, the resource name is not defined to RACF and DB2 rejects the request (with reason code X'00F30013'). For instructions on defining the resource name, see “Defining DB2 resources to RACF” on page 200.

   **Not = 4**, RACF is not active. DB2 continues with the next step, but the connection request and the user are not verified.

   **If RC = 0**, RACF is active and has verified the RACF user ID; DB2 continues with the next step.

3. DB2 runs the connection exit routine. To use DB2 secondary IDs, you must replace the exit routine. See “Supplying secondary IDs for connection requests” on page 172.

   If you *do not* want to use secondary IDs, do nothing. The IBM-supplied default connection exit routine continues the connection processing. The processing has the following effects:

   - If a value for the initial primary authorization ID exists, the value becomes the DB2 primary ID.
   - If no value exists (the value is blank), the primary ID is set by default, as shown in Table 51 on page 172.
   - The SQL ID is set equal to the primary ID.
   - No secondary IDs exist.

   If you want to use secondary IDs, see the description in “Supplying secondary IDs for connection requests” on page 172. Of course, you can also replace the exit routine with one that provides different default values for the DB2 primary ID. If you have written such a routine for an earlier release of DB2, it will probably work for this release with no change.

*Table 51. Sources of default authorization identifiers*

| Source | Default primary authorization ID |
|---|---|
| TSO | TSO logon ID |
| BATCH | USER parameter on JOB statement |
| Started task, or batch job with no USER parameter | Default authorization ID set when DB2 was installed (UNKNOWN AUTHID on installation panel DSNTIPP) |
| Remote request | None. The user ID is required and is provided by the DRDA requester. |

## Supplying secondary IDs for connection requests

If you want to use DB2 secondary authorization IDs, you must replace the default connection exit routine. If you want to use RACF group names as DB2 secondary IDs, as illustrated in “Examples of granting and revoking privileges” on page 140, the easiest method is to use the IBM-supplied sample routine.

Distinguish those two routines carefully.

- The *default* connection exit routine is supplied as object code, is installed as part of the normal procedure for installing DB2, and provides values only for the DB2 primary and SQL IDs—not for secondary IDs.
- The *sample* connection exit routine is supplied as source code (you can change it), must be compiled and placed in a DB2 library, and provides for secondary IDs, primary IDs, and SQL IDs. Installation job DSNTIJEX replaces the default connection exit routine with the sample connection exit routine; see Part 2 of *DB2 Installation Guide* for more information.

The sample connection exit routine has the following effects:

- The sample connection exit routine sets the DB2 primary ID the same way as it is set by the default routine. If the initial primary ID is not blank, it becomes the DB2 primary ID. If the initial primary ID is blank, the sample routine provides the same default value as does the default routine. If the sample routine cannot find a nonblank primary ID, DB2 uses the default ID (UNKNOWN AUTHID) from installation panel DSNTIPP. In that case, no secondary IDs are supplied.
- If the connection request is from a TSO-managed address space, the routine sets the SQL ID to the TSO data set name prefix in the TSO user profile table, but only if the TSO data set name prefix is also equal to the primary ID or one of the secondary IDs. Those requests include requests through the call attachment facility, and requests from TSO foreground and background. In all other cases, the routine sets the SQL ID equal to the primary ID.
- The secondary authorization IDs depend on RACF options:
  - If RACF is not active, no secondary IDs exist.
  - If RACF is active but its “list of groups” option is not active, one secondary ID exists (the default connected group name) if that was supplied by the attachment facility.
  - If RACF is active and you selected the “list of groups” option, the routine sets the list of DB2 secondary IDs to the list of group names to which the RACF user ID is connected (but not in REVOKE status). The maximum number of groups is 245. The list of group names is obtained from RACF and includes the default connected group name.

If you need something that is not provided by either the default or the sample connection exit routine, you can write your own routine. For instructions, see “Appendix B. Writing exit routines” on page 901.

## Required CICS specifications

In order for a CICS transaction to use the sample connection or sign-on exit routines, the external security system, such as RACF, must be defined to CICS with these specifications:

- The CICS system initialization table must specify external security. For CICS Version 4, specify SEC=YES; for earlier releases of CICS, specify EXTSEC=YES. If you are using the CICS multiple region option (MRO), you must specify SEC=YES or EXTSEC=YES for every CICS system that is connected by interregion communication (IRC).
- If your version of CICS uses a sign-on table (SNT), the CICS sign-on table must specify EXTSEC=YES for each signed on user that uses the sign-on exit.
- When the user signs on to a CICS terminal-owning region, the terminal-owning region must propagate the authorization ID to the CICS application-owning region. For more information on that propagation, see the description of ATTACHSEC in the applicable version of the CICS *Intercommunication Guide*.

You must change the sample sign-on exit routine (DSN3SSGN) before using it if the following conditions are all true. For instructions, see “Sample exit routines” on page 902.

- You attach to DB2 with an AUTH parameter in the RCT other than AUTH=GROUP.
- You have the RACF list-of-groups option active.
- You have transactions whose initial primary authorization ID is not defined to RACF.

# Processing sign-ons

For requests from IMS dependent regions, CICS transaction subtasks, or OS/390 RRS connections, the initial primary ID is not obtained until just before allocating a plan for a transaction. A new sign-on request can run the same plan without deallocating the plan and reallocating it. Nevertheless, the new sign-on request can change the primary ID.

Unlike connection processing, sign-on processing does not check the RACF user ID of the address space. The steps are shown in Figure 16.

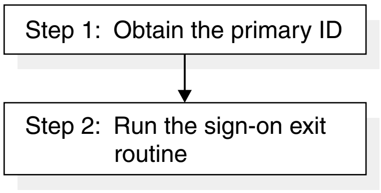

Step 1: Obtain the primary ID
Step 2: Run the sign-on exit routine

*Figure 16. Sign-on processing*

## The steps in detail

DB2 takes the following steps in processing sign-ons:

1. Determine the initial primary ID as follows:

**For IMS sign-ons** from message-driven regions, if the user has signed on, the initial primary authorization ID is the user's sign-on ID.

IMS passes to DB2 the IMS sign-on ID and the associated RACF connected group name, if one exists.

If the user has not signed on, the primary ID is the LTERM name, or if that is not available, the PSB name.

For a batch-oriented region, the primary ID is the value of the USER parameter on the job statement, if that is available. If that is not available, the primary ID is the program's PSB name.

**For CICS sign-ons**, the initial primary authorization ID is specified by authorization directives in the CICS resource control table (RCT). For instructions on setting up the RCT to indicate the appropriate ID, see the description of the AUTH option in the macro DSNCRCT TYPE=ENTRY in Part 2 of *DB2 Installation Guide* , and also the information there about coordinating CICS and DB2 security.

You can use the following values for authorization IDs:

- The VTAM application name for the CICS system; use AUTH=SIGNID.
- A character string up to eight characters long, which is supplied in the RCT; use AUTH=(*string*).
- The CICS group ID (eight characters); use AUTH=GROUP. That option passes to DB2 the CICS user ID and the associated RACF connected group name. AUTH=GROUP is not a valid authorization type for transactions that do not have RACF user IDs that are associated with them (for example, non-terminal-driven transactions in releases of CICS before CICS Version 4).
- The CICS user ID (eight characters); use AUTH=USERID. AUTH=USERID is not a valid authorization type for transactions that do not have signed-on user IDs that are associated with them (for example, non-terminal-driven transactions in releases of CICS before CICS Version 4).
- The operator ID (three characters padded on the right with five blanks); use AUTH=USER. AUTH=USER is valid only for transactions that are associated with a signed-on USERID or a terminal.
- The terminal ID (four characters padded with four blanks); use AUTH=TERM. AUTH=TERM is valid only for transactions associated with a terminal.
- The transaction ID (four characters padded with four blanks); use AUTH=TXID.

**For remote requests**, the source of the initial primary ID is determined by entries in the SYSIBM.USERNAMES table. “Accepting a remote attachment request” on page 180 explains how to control the ID.

**For connections using Recoverable Resource Manager Services attachment facility**, the processing depends on the type of signon request:

- SIGNON
- AUTH SIGNON
- CONTEXT SIGNON

For SIGNON, the primary authorization ID is retrieved from ACEEUSRI if an ACEE is associated with the TCB (TCBSENV). This is the normal case. However, if an ACEE is not associated with the TCB, SIGNON uses the primary authorization ID that is associated with the address space, that is, from the ASXB. If the new primary authorization ID was retrieved from the ACEE that is associated with the TCB and ACEEGRPN is not null, DB2 uses ACEEGRPN to establish secondary authorization IDs.

With AUTH SIGNON, an APF-authorized program can pass a primary authorization ID for the connection. If a primary authorization ID is passed, AUTH SIGNON also uses the value that is passed in the secondary authorization ID parameter to establish secondary authorization IDs. If the primary authorization ID is not passed, but a valid ACEE is passed, AUTH SIGNON uses the value in ACEEUSRI for the primary authorization ID if ACEEUSRL is not 0. If ACEEUSRI is used for the primary authorization ID, AUTH SIGNON uses the value in ACEEGRPN as the secondary authorization ID if ACEEGRPL is not 0.

For CONTEXT SIGNON, the primary authorization ID is retrieved from data that is associated with the current RRS context using the context_key, which is supplied as input. CONTEXT SIGNON uses the CTXSDTA and CTXRDTA functions of RRS context services. An authorized function must use CTXSDTA to store a primary authorization ID prior to invoking CONTEXT SIGNON. Optionally, CTXSDTA can be used to store the address of an ACEE in the context data that has a context_key that was supplied as input to CONTEXT SIGNON. DB2 uses CTXRDTA to retrieve context data. If an ACEE address is passed, CONTEXT SIGNON uses the value in ACEEGRPN as the secondary authorization ID if ACEEGRPL is not 0.

For more information, see Part 6 of *DB2 Application Programming and SQL Guide*.

2. DB2 runs the sign-on exit routine. **User action:** To use DB2 secondary IDs, you must replace the exit routine.

   If you *do not* want to use secondary IDs, do nothing. Sign-on processing is then continued by the IBM-supplied default sign-on exit routine, which has the following effects:
   - The initial primary authorization ID remains the primary ID.
   - The SQL ID is set equal to the primary ID.
   - No secondary IDs exist.

   You can replace the exit routine with one of your own, even if it has nothing to do with secondary IDs. If you do, remember that IMS and CICS recovery coordinators, their dependent regions, and RRSAF take the exit routine only if they have provided a user ID in the sign-on parameter list.

   If you *do* want to use secondary IDs, see the description that follows.

## Supplying secondary IDs for sign-on requests

If you want the primary authorization ID to be associated with DB2 secondary authorization IDs, you must replace the default sign-on exit routine. The procedure is like that for connection processing: If you want to use RACF group names as DB2 secondary IDs, the easiest method is to use the IBM-supplied sample routine. An installation job can automatically replace the default routine with the sample routine; to run it, see “Installation Step 6: Define User Authorization Exit Routines: DSNTIJEX” in Part 2 of *DB2 Installation Guide*.

Distinguish carefully between the two routines. The default sign-on routine provides no secondary IDs and has the effects described in step 2 of “Processing sign-ons” on page 173. The sample sign-on routine supports DB2 secondary IDs, and is like the sample connection routine.

The sample sign-on routine has the following effects:

- The initial primary authorization ID is left unchanged as the DB2 primary ID.

- The SQL ID is made equal to the DB2 primary ID.
- The secondary authorization IDs depend on RACF options:
  - If RACF is not active, no secondary IDs exist.
  - If RACF is active but its “list of groups” option is not active, one secondary ID exists; it is the name passed by CICS or by IMS.
  - If RACF is active and you have selected the option for a list of groups, the routine sets the list of DB2 secondary IDs to the list of group names to which the RACF user ID is connected, up to a limit of 245 groups. The list of group names includes the default connected group name.

# Controlling requests from remote applications

If you are controlling requests from remote applications, your DB2 subsystem might be accepting requests from applications that use SNA network protocols, TCP/IP network protocols, or both. This section describes the methods that the DB2 server can use to control access from those applications. To understand what is described here, you must be familiar with the communications database, which is part of the DB2 catalog. The following topics are described in this section:

- “Overview of security mechanisms for DRDA and SNA”
- “The communications database for the server” on page 178
- “Controlling inbound connections that use SNA protocols” on page 180
- “Controlling inbound connections that use TCP/IP protocols” on page 187

## Overview of security mechanisms for DRDA and SNA

SNA and DRDA have different security mechanisms. DRDA lets a user be authenticated using SNA security mechanisms or DRDA mechanisms, which are independent of the underlying network protocol. For an SNA network connection, a DRDA requester can send security tokens using a SNA attach or using DRDA commands. DB2 for OS/390 and z/OS as a requester uses SNA security mechanisms if it uses a SNA network connection (except for Kerberos) and DRDA security mechanisms for TCP/IP network connections (or when Kerberos authentication is chosen, regardless of the network type).

### Mechanisms used by DB2 for OS/390 and z/OS as a requester

DB2 for OS/390 and z/OS as a requester chooses SNA or DRDA security mechanisms based on the network protocol and the authentication mechanisms you use. If you use SNA protocols, the following SNA authentication mechanisms are supported:

- User ID only (already verified)
- User ID and password, described in “Sending passwords” on page 197
- User ID and PassTicket, described in “Sending RACF PassTickets” on page 197

Authentication is performed based on SNA protocols, which means that the authentication tokens are sent in an SNA attach (FMH-5).

If you use TCP/IP protocols, the following DRDA authentication mechanisms are supported:

- User ID only (already verified)
- User ID and password, described in “Sending passwords” on page 197
- User ID and PassTicket, described in “Sending RACF PassTickets” on page 197

Authentication is performed based on DRDA protocols, which means that the authentication tokens are sent in DRDA security flows.

If you use a requester other than DB2 for OS/390 and z/OS, refer to that product's documentation.

## Mechanisms accepted by DB2 for OS/390 and z/OS as a server

DB2 for OS/390 and z/OS as a server can accept either SNA or DRDA authentication mechanisms. This means that DB2 can authenticate remote users from either the security tokens obtained from the SNA ATTACH (FMH-5) or from the DRDA security commands described by each of the protocols. The following authentication methods are supported by DB2 for OS/390 and z/OS as a server:

- User ID only (already verified at the requester)
- User ID and password, described in “Sending passwords” on page 197
- User ID and PassTicket, described in “Sending RACF PassTickets” on page 197
- Kerberos tickets, described in “Establishing Kerberos authentication through RACF” on page 212
- Unencrypted user ID and encrypted password, described in “Sending encrypted passwords from a workstation” on page 198
- Encrypted user ID and encrypted password, described in “Sending encrypted passwords from a workstation” on page 198
- User ID, password, and new password, described in “Allowing users to change expired passwords”

***Allowing users to change expired passwords:*** DB2 can return to the DRDA requester information about errors and expired passwords. To allow this, specify YES in the EXTENDED SECURITY field of installation panel DSNTIPR.

When the DRDA requester is notified that the RACF password has expired, and the requester has implemented function to allow passwords to be changed, the requester can prompt the end user for the old password and a new password. The requester sends the old and new passwords to the DB2 server. This function is supported through DB2 Connect.

With the extended security option, DB2 passes the old and new passwords to RACF. If the old password is correct, and the new password meets the installation's password requirements, the end user's password is changed and the DRDA connection request is honored.

When a user changes a password, the user ID, the old password, and the new password are sent to DB2. You can now encrypt these three tokens before they are sent from the client.

***Detecting authorization failures (EXTENDED SECURITY):*** If the DB2 server is installed with YES for the EXTENDED SECURITY field of installation panel DSNTIPR, detailed reason codes are returned to a DRDA client when a DDF connection request fails because of security errors. When using SNA protocols, the requester must have included support for extended security sense codes. One such product is DB2 Connect.

If the proper requester support is present, the requester generates SQLCODE -30082 (SQLSTATE '08001') with a specific indication for the failure. Otherwise, a generic security failure code is returned.

## The communications database for the server

The information in this section, up to “Controlling inbound connections that use SNA protocols” on page 180, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

The communications database (CDB) is a set of DB2 catalog tables that let you control aspects of how requests leave this DB2 and how requests come in. This section concentrates on the columns of the communications database that pertain to security on the inbound side (the server).

The SYSIBM.IPNAMES table is not described in this section, because that table is not used to control inbound TCP/IP requests.

### Columns used in SYSIBM.LUNAMES

This table is used only for requests that use SNA protocols.

**LUNAME CHAR(8)**

The LUNAME of the remote system. A blank value identifies a default row that serves requests from any system that is not specifically listed elsewhere in the column.

**SECURITY_IN CHAR(1)**

The *acceptance option* for a remote request from the corresponding LUNAME:

**V** The option is “verify.” An incoming request must include one of the following authentication entities:

- User ID and password
- User ID and RACF PassTicket, described in “Sending RACF PassTickets” on page 197
- User ID and RACF encrypted password (not recommended)
- Kerberos security tickets, described in “Establishing Kerberos authentication through RACF” on page 212
- User ID and DRDA encrypted password, described in “Sending encrypted passwords from a workstation” on page 198.
- User ID, password, and new password, described in “Allowing users to change expired passwords” on page 177
- User ID and encrypted password, or encrypted user ID and encrypted password, described in “Allowing users to change expired passwords” on page 177

**A** The option is “already verified.” This is the default. With A, a request does not need an authentication token, although the token is checked if it is sent.

With this option, an incoming connection request is accepted if it includes any of the following authentication tokens:

- User ID only
- All authentication methods that option V supports

If the USERNAMES column of SYSIBM.LUNAMES contains I or B, RACF is not invoked to validate incoming connection requests that contain only a user ID.

**ENCRYPTPSWDS CHAR(1)**

This column only applies to DB2 for OS/390 and z/OS or DB2 for MVS/ESA partners when passwords are used as authentication tokens. It indicates whether passwords received from and sent to the corresponding LUNAME are encrypted:

**Y** Yes, passwords are encrypted. For outbound requests, the encrypted password is extracted from RACF and sent to the server. For inbound requests, the password is treated as if it is encrypted.

**N** No, passwords are not encrypted. This is the default; any character other than Y is treated as N. Specify N for CONNECT statements that contain a USER parameter.

**Recommendation**: When you connect to a DB2 for OS/390 and z/OS partner that is at Version 5 or a subsequent release, use RACF PassTickets (SECURITY_OUT='R') instead of using passwords.

**USERNAMES CHAR(1)**

This column indicates whether an ID accompanying a remote request, sent from or to the corresponding LUNAME, is subject to translation and “come from” checking. When you specify I, O, or B, use the SYSIBM.USERNAMES table to perform the translation.

**I** An inbound ID is subject to translation.

**O** An outbound ID, sent to the corresponding LUNAME, is subject to translation.

**B** Both inbound and outbound IDs are subject to translation.

**blank** No IDs are translated.

## Columns used in SYSIBM.USERNAMES

This table is used by both SNA and TCP/IP connections.

**TYPE CHAR(1)**

Indicates whether the row is used for inbound or outbound translation:

**I** The row applies to inbound IDs (not applicable for TCP/IP connections).

**O** The row applies to outbound IDs.

The field should contain only I or O. Any other character, including blank, causes the row to be ignored.

**AUTHID CHAR(8)**

An authorization ID that is permitted and perhaps translated. If blank, any authorization ID is permitted with the corresponding LINKNAME; all authorization IDs are translated in the same way. Outbound translation is not performed on CONNECT statements that contain an authorization ID for the value of the USER parameter.

**LINKNAME CHAR(8)**

Identifies the VTAM or TCP/IP network locations that are associated with this row. A blank value in this column indicates that this name translation rule applies to any TCP/IP or SNA partner.

If you specify a nonblank value for this column, one or both of the following situations must be true:

- A row exists in table SYSIBM.LUNAMES that has an LUNAME value that matches the LINKNAME value that appears in this column.
- A row exists in table SYSIBM.IPNAMES that has a LINKNAME value that matches the LINKNAME value that appears in this column.

**NEWAUTHID CHAR(8)**

The translated authorization ID. If blank, no translation occurs.

## Controlling inbound connections that use SNA protocols

Requests from a remote LU are subject to two security checks before they come into contact with DB2. Those checks control what LUs can attach to the network and verify the identity of a partner LU.

Finally, DB2 itself imposes several checks before accepting an attachment request.

If using private protocols, the LOCATIONS table controls the locations that can access DB2. To allow a remote location to access DB2, the remote location name must be specified in the SYSIBM.LOCATIONS table. This check is only supported for connections using private protocols.

### Controlling what LUs can attach to the network

This check is carried out by VTAM, to prevent an unauthorized LU from attaching to the network and presenting itself to other LUs as an acceptable partner in communication. It requires each LU that attaches to the network to identify itself by a password. If that requirement is in effect for your network, your DB2 subsystem, like every other LU on the network, must:

1. Choose a VTAM password.
2. Code the password with the PRTCT parameter of the VTAM APPL statement, when you define your DB2 to VTAM. The APPL statement is described in detail in Part 3 of *DB2 Installation Guide*.

### Verifying a partner LU

This check is carried out by RACF and VTAM, to check the identity of an LU sending a request to your DB2. **Recommendation**: Specify partner-LU verification, which requires the following steps:

1. Code VERIFY=REQUIRED on the VTAM APPL statement, when you define your DB2 to VTAM. The APPL statement is described in detail in Part 3 of *DB2 Installation Guide*.
2. Establish a RACF profile for each LU from which you permit a request. For the steps required, see “Enable partner-LU verification” on page 202.

### Accepting a remote attachment request

When VTAM has established a conversation for a remote application, that application sends a *remote request*, which is a request to attach to your local DB2. (Do not confuse the *remote* request with a *local* attachment request that comes through one of the DB2 attachment facilities—IMS, CICS, TSO, and so on. A remote attachment request is defined by Systems Network Architecture and LU 6.2 protocols; specifically, it is an SNA Function Management Header 5.)

This section tells what security checks you can impose on remote attachment requests.

***Conversation-level security:*** This section assumes that you have defined your DB2 to VTAM with the conversation-level security set to “already verified”. (To do that, you coded SECACPT=ALREADYV on the VTAM APPL statement, as described in Part 3 of *DB2 Installation Guide*. That value provides more options than does “conversation” (SECACPT=CONV), which we do not recommend.

***Steps, tools, and decisions:*** The steps an attachment request goes through before acceptance allow much flexibility in choosing security checks. Scan Figure 17 on page 183 to see what is possible.

The primary tools for controlling remote attachment requests are entries in tables SYSIBM.LUNAMES and SYSIBM.USERNAMES in the communications database. You need a row in SYSIBM.LUNAMES for each system that sends attachment requests, a dummy row that allows *any* system to send attachment requests, or both. You might need rows in SYSIBM.USERNAMES to permit requests from specific IDs or specific LUNAMES, or to provide translations for permitted IDs.

When planning to control remote requests, answer the questions posed by the following topics for each remote LU that can send a request.

1. "Do you permit access?"
2. "Do you manage inbound IDs through DB2 or RACF?"
3. "Do you trust the partner LU?"
4. "If you use passwords, are they encrypted?" on page 182
5. "Do you translate inbound IDs?" on page 185
6. "How do you associate inbound IDs with secondary IDs?" on page 186

***Do you permit access?:*** To permit attachment requests from a particular LU, you need a row in your SYSIBM.LUNAMES table. The row must either give the specific LUNAME or it must be a dummy row with the LUNAME blank. (The table can have only one dummy row, which is used by all LUs for which no specific row exists, when making requests.) Without one of those rows, the attachment request is rejected.

***Do you manage inbound IDs through DB2 or RACF?:*** If you manage incoming IDs through RACF, you must register every acceptable ID with RACF, and DB2 must call RACF to process every request. If you manage incoming IDs through RACF, either RACF or Kerberos can be used to authenticate the user. Kerberos cannot be used if you do not have RACF on the system.

If you manage incoming IDs through DB2, you can avoid calls to RACF and can specify acceptance of many IDs by a single row in the SYSIBM.USERNAMES table.

To manage incoming IDs through DB2, put an I in the USERNAMES column of SYSIBM.LUNAMES for the particular LU. (Or, if an O is there already because you are also sending requests *to* that LU, change O to B.) Attachment requests from that LU now go through sign-on processing, and its IDs are subject to translation. (For more information about translating IDs, see "Do you translate inbound IDs?" on page 185.)

To manage incoming IDs through RACF, leave USERNAMES blank for that LU (or leave the O unchanged). Requests from that LU go through connection processing, and its IDs are not subject to translation.

***Do you trust the partner LU?:*** Presumably, RACF has already validated the identity of the other LU (described in "Verifying a partner LU" on page 180). If you trust incoming IDs from that LU, you do not need to validate them by an authentication token. Put an A in the SECURITY_IN column of the row in SYSIBM.LUNAMES that corresponds to the other LU; your acceptance level for requests from that LU is now "already verified". Requests from that LU are accepted without an authentication token. (In order to use this option, you must have defined DB2 to VTAM with SECACPT=ALREADYV, as described in 180.) If an authentication token *does* accompany a request, DB2 calls RACF to check the authorization ID against it. To require an authentication token from a particular LU, put a V in the SECURITY_IN column in SYSIBM.LUNAMES; your acceptance level for requests from that LU is now “verify”. You must also register every acceptable incoming ID and its password with RACF.

***Performance considerations:*** Each request to RACF to validate authentication tokens results in an I/O operation, which has a high performance cost.

**Recommendation:** To eliminate the I/O, allow RACF to cache security information in VLF. To activate this option, add the IRRACEE class to the end of MVS VLF member COFVLFxx in SYS1.PARMLIB, as follows:

```
CLASS NAME(IRRACEE)
EMAJ (ACEE)
```

***If you use passwords, are they encrypted?:*** Passwords can be encrypted through:

- RACF using PassTickets, described in “Sending RACF PassTickets” on page 197.
- DRDA password encryption support. DB2 for OS/390 and z/OS as a server supports DRDA encrypted passwords and encrypted user IDs with encrypted passwords. See “Sending encrypted passwords from a workstation” on page 198 for more information.

***If you use Kerberos, are users authenticated?:*** If your distributed environment uses Kerberos to manage users and perform user authentication, DB2 for OS/390 and z/OS can use Kerberos security services to authenticate remote users. See “Establishing Kerberos authentication through RACF” on page 212.

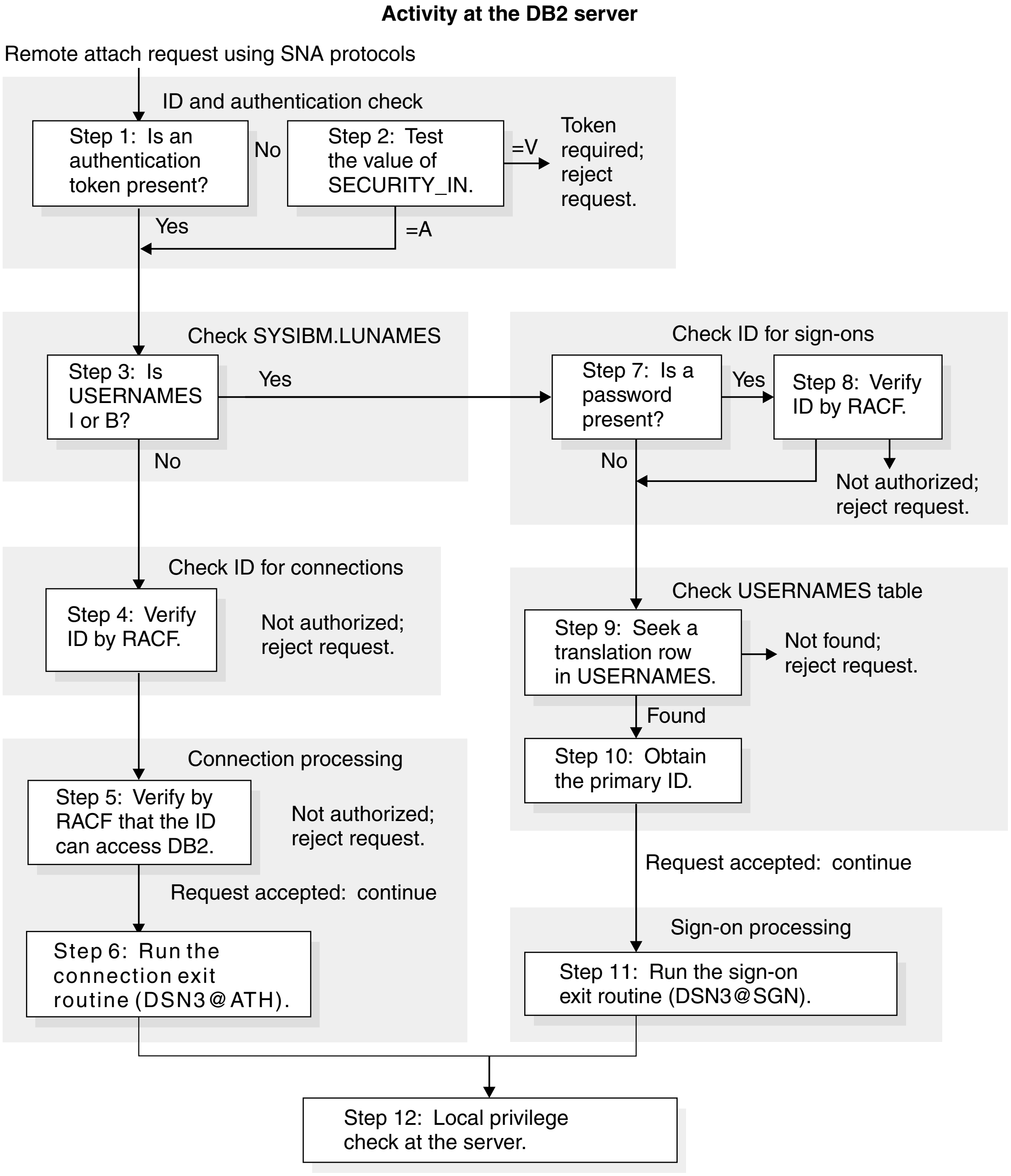

Activity at the DB2 server
Remote attach request using SNA protocols
ID and authentication check
Step 1: Is an authentication token present?
No
Step 2: Test the value of SECURITY_IN.
=V
Token required; reject request.
Yes
=A
Check SYSIBM.LUNAMES
Step 3: Is USERNAMES I or B?
Yes
No
Check ID for sign-ons
Step 7: Is a password present?
Yes
Step 8: Verify ID by RACF.
No
Not authorized; reject request.
Check ID for connections
Step 4: Verify ID by RACF.
Not authorized; reject request.
Check USERNAMES table
Step 9: Seek a translation row in USERNAMES.
Not found; reject request.
Found
Step 10: Obtain the primary ID.
Connection processing
Step 5: Verify by RACF that the ID can access DB2.
Not authorized; reject request.
Request accepted: continue
Step 6: Run the connection exit routine (DSN3@ATH).
Request accepted: continue
Sign-on processing
Step 11: Run the sign-on exit routine (DSN3@SGN).
Step 12: Local privilege check at the server.

*Figure 17. Steps in accepting a remote attachment request from requester that is using SNA*

***Details of remote attachment request processing:***

1. If the remote request has no authentication token, DB2 checks the security acceptance option in the SECURITY_IN column of table SYSIBM.LUNAMES. No password is sent or checked for the plan or package owner that is sent from a DB2 subsystem.
2. If the acceptance option is “verify” (SECURITY_IN = V), a security token is required to authenticate the user. DB2 rejects the request if the token missing.
3. If the USERNAMES column of SYSIBM.LUNAMES contains I or B, the authorization ID, and the plan or package owner that is sent by a DB2

subsystem, are subject to translation under control of the SYSIBM.USERNAMES table. If the request is allowed, it eventually goes through sign-on processing.

If USERNAMES does not contain I or B, the authorization ID is not translated.

4. DB2 calls RACF by the RACROUTE macro with REQUEST=VERIFY to check the ID. DB2 uses the PASSCHK=NO option if no password is specified and ENCRYPT=YES if the ENCRYPTPSWDS column of SYSIBM.LUNAMES contains Y. If the ID, password, or PassTicket cannot be verified, DB2 rejects the request.

   In addition, depending on your RACF environment, the following RACF checks may also be performed:

   - If the RACF APPL class is active, RACF verifies that the ID has been given access to the DB2 APPL. The APPL resource that is checked is the LU name that the requester used when the attachment request was issued. This is either the local DB2 LU name or the generic LU name.
   - If the RACF APPCPORT class is active, RACF verifies that the ID is authorized to access MVS from the port of entry (POE). The POE that is use in the verify call is the requesting LU name.

5. The remote request is now treated like a local connection request with a DIST environment for the DSNR resource class; for details, see “Processing connections” on page 170. DB2 calls RACF by the RACROUTE macro with REQUEST=AUTH, to check whether the authorization ID is allowed to use DB2 resources that are defined to RACF.

   The RACROUTE macro call also verifies that the user is authorized to use DB2 resources from the requesting system, known as the port of entry (POE); for details, see “Allowing access from remote requesters” on page 208.

6. DB2 invokes the connection exit routine. The parameter list that is passed to the routine describes where a remote request originated.

7. If no password exists, RACF is not called. The ID is checked in SYSIBM.USERNAMES.

8. If a password exists, DB2 calls RACF through the RACROUTE macro with REQUEST=VERIFY to verify that the ID is known with the password. ENCRYPT=YES is used if the ENCRYPTPSWDS column of SYSIBM.LUNAMES contains Y. If DB2 cannot verify the ID or password, the request is rejected.

9. DB2 searches SYSIBM.USERNAMES for a row that indicates how to translate the ID. The need for a row that applies to a particular ID and sending location imposes a “come-from” check on the ID: If no such row exists, DB2 rejects the request.

10. If an appropriate row is found, DB2 translates the ID as follows:
    - If a nonblank value of NEWAUTHID exists in the row, that value becomes the primary authorization ID.
    - If NEWAUTHID is blank, the primary authorization ID remains unchanged.

11. The remote request is now treated like a local sign-on request; for details, see “Processing sign-ons” on page 173. DB2 invokes the sign-on exit routine. The parameter list that is passed to the routine describes where a remote request originated. For details, see “Connection and sign-on routines” on page 901.

12. The remote request now has a primary authorization ID, possibly one or more secondary IDs, and an SQL ID. A request from a remote DB2 is also known by a plan or package owner. Privileges and authorities that are granted to those IDs at the DB2 server govern the actions that the request can take.

***Do you translate inbound IDs?:*** Ideally, each of your authorization IDs has the same meaning throughout your entire network. In practice, that might not be so, and the duplication of IDs on different LUs is a security exposure. For example, suppose that the ID DBADM1 is known to the local DB2 and has DBADM authority over certain databases there; suppose also that the same ID exists in some remote LU. If an attachment request comes in from DBADM1, and if nothing is done to alter the ID, the wrong user can exercise privileges of DBADM1 in the local DB2. The way to protect against that exposure is to translate the remote ID into a different ID before the attachment request is accepted.

You must be prepared to translate the IDs of plan owners, package owners, and the primary IDs of processes that make remote requests. For the IDs that are sent to you by other DB2 LUs, see "What IDs you send" on page 193. (Do not plan to translate all IDs in the connection exit routine—the routine does not receive plan and package owner IDs.)

If you have decided to manage inbound IDs through DB2, you can translate an inbound ID to some other value. Within DB2, you grant privileges and authorities only to the translated value. As Figure 17 on page 183 shows, that "translation" is not affected by anything you do in your connection or sign-on exit routine. The *output* of the translation becomes the *input* to your sign-on exit routine. ***Recommendation:*** Do not translate inbound IDs in an exit routine; translate them only through the SYSIBM.USERNAMES table.

The examples in Table 52 shows the possibilities for translation and how to control translation by SYSIBM.USERNAMES. You can use entries to allow requests only from particular LUs or particular IDs, or from combinations of an ID and an LU. You can also translate any incoming ID to another value. Table 53 on page 186 shows the search order of the SYSIBM.USERNAMES table.

***Performance considerations:*** In the process of accepting remote attachment requests, any step that calls RACF is likely to have a relatively high performance cost. To trade some of that cost for a somewhat greater security exposure, have RACF check the identity of the other LU just once, as described under "Verifying a partner LU" on page 180. Then trust the partner LU, translating the inbound IDs and not requiring or using passwords. In this case, no calls are made to RACF from within DB2; the penalty is only that you make the partner LU responsible for verifying IDs.

***Update considerations:*** If you update tables in the CDB while the distributed data facility is running, the changes might not take effect immediately. For details, see Part 3 of *DB2 Installation Guide*.

**Example:** Table 52 shows how USERNAMES translates inbound IDs.

*Table 52. Your SYSIBM.USERNAMES table. (Row numbers are added for reference.)*

| Row | TYPE | AUTHID | LINKNAME | NEWAUTHID |
|---|---|---|---|---|
| 1 | I | blank | LUSNFRAN | blank |
| 2 | I | BETTY | LUSNFRAN | ELIZA |
| 3 | I | CHARLES | blank | CHUCK |
| 4 | I | ALBERT | LUDALLAS | blank |
| 5 | I | BETTY | blank | blank |

DB2 searches SYSIBM.USERNAMES to determine how to translate for each of the following requests:

ALBERT requests from LUDALLAS
DB2 searches for an entry for AUTHID=ALBERT and LINKNAME=LUDALLAS. DB2 finds one in row 4, so the request is accepted. The value of NEWAUTHID in that row is blank, so ALBERT is left unchanged.

BETTY requests from LUDALLAS
DB2 searches for an entry for AUTHID=BETTY and LINKNAME=LUDALLAS; none exists. DB2 then searches for AUTHID=BETTY and LINKNAME=blank. It finds that entry in row 5, so the request is accepted. The value of NEWAUTHID in that row is blank, so BETTY is left unchanged.

CHARLES requests from LUDALLAS
DB2 searches for AUTHID=CHARLES and LINKNAME=LUDALLAS; no such entry exists. DB2 then searches for AUTHID=CHARLES and LINKNAME=blank. The search ends at row 3; the request is accepted. The value of NEWAUTHID in that row is CHUCK, so CHARLES is translated to CHUCK.

ALBERT requests from LUSNFRAN
DB2 searches for AUTHID=ALBERT and LINKNAME=LUSNFRAN; no such entry exists. DB2 then searches for AUTHID=ALBERT and LINKNAME=blank; again no entry exists. Finally, DB2 searches for AUTHID=blank and LINKNAME=LUSNFRAN, finds that entry in row 1, and the request is accepted. The value of NEWAUTHID in that row is blank, so ALBERT is left unchanged.

BETTY requests from LUSNFRAN
DB2 finds row 2, and BETTY is translated to ELIZA.

CHARLES requests from LUSNFRAN
DB2 finds row 3 before row 1; CHARLES is translated to CHUCK.

WILBUR requests from LUSNFRAN
No provision is made for WILBUR, but row 1 of the SYSIBM.USERNAMES table allows any ID to make a request from LUSNFRAN and to pass without translation. The acceptance level for LUSNFRAN is “already verified”, so WILBUR can pass without a password check by RACF. After accessing DB2, WILBUR can use only the privileges that are granted to WILBUR and to PUBLIC (for DRDA access) or to PUBLIC AT ALL LOCATIONS (for DB2 private-protocol access).

WILBUR requests from LUDALLAS
Because the acceptance level for LUDALLAS is “verify” as recorded in the SYSIBM.LUNAMES table, WILBUR must be known to the local RACF. DB2 searches in succession for one of the combinations WILBUR/LUDALLAS, WILBUR/blank, or blank/LUDALLAS. None of those is in the table, so the request is rejected. The absence of a row permitting WILBUR to request from LUDALLAS imposes a “come-from” check: WILBUR can attach from some locations (LUSNFRAN), and some IDs (ALBERT, BETTY, and CHARLES) can attach from LUDALLAS, but WILBUR cannot attach if coming from LUDALLAS.

Table 53 shows the search order for the SYSIBM.USERNAMES table:

*Table 53. Precedence search order for SYSIBM.USERNAMES table*

| AUTHID | LINKNAME | Result |
|---|---|---|
| Name | Name | If NEWAUTHID is specified, AUTHID is translated to NEWAUTHID for the specified LINKNAME. |
| Name | Blank | If NEWAUTHID is specified, AUTHID is translated to NEWAUTHID for all LINKNAMEs. |
| Blank | Name | If NEWAUTHID is specified, it is substituted for AUTHID for the specified LINKNAME. |
| Blank | Blank | Unavailable resource message (SQLCODE -904) is returned. |

***How do you associate inbound IDs with secondary IDs?:*** Your decisions on the previous questions determine what value is used for the primary authorization

ID on an attachment request. They also determine whether those requests are next treated as connection requests or as sign-on requests. That means that the remote request next goes through the same processing as a local request, and that you have the opportunity to associate the primary ID with a list of secondary IDs in the same way you do for local requests. For more information about processing connections and sign-ons, see “Processing connections” on page 170 and “Processing sign-ons” on page 173.

## Controlling inbound connections that use TCP/IP protocols

DRDA connections that use TCP/IP have fewer security controls than do connections that use SNA protocols. When planning to control inbound TCP/IP connections, consider the following issues:

***Do you permit access by TCP/IP?*** If the serving DB2 for OS/390 and z/OS subsystem has a DRDA port and resynchronization port specified in the BSDS, DB2 is enabled for TCP/IP connections.

***Do you manage inbound IDs through DB2 or RACF?*** All IDs must be passed to RACF or Kerberos for processing. No option exists to handle incoming IDs through DB2.

***Do you trust the partner?*** TCP/IP does not verify partner LUs as SNA does. If your requesters support mutual authentication, use Kerberos to handle this on the requester side.

***If you use passwords, are they encrypted?*** Passwords can be encrypted through:

- RACF using PassTickets, described in “Sending RACF PassTickets” on page 197.
- DRDA password encryption support. DB2 for OS/390 and z/OS as a server supports DRDA encrypted passwords and encrypted user IDs with encrypted passwords. See “Sending encrypted passwords from a workstation” on page 198 for more information.

***If you use Kerberos, are users authenticated?*** If your distributed environment uses Kerberos to manage users and perform user authentication, DB2 for OS/390 and z/OS can use Kerberos security services to authenticate remote users. See “Establishing Kerberos authentication through RACF” on page 212.

***Do you translate inbound IDs?*** Inbound IDs are not translated when you use TCP/IP.

***How do you associate inbound IDs with secondary IDs?*** To associate an inbound ID with secondary IDs, modify the default connection exit routine (DSN3@ATH). TCP/IP requests do not use the sign-on exit routine.

### Steps, tools, and decisions

See Figure 18 on page 188 for an overview of how incoming requests are handled. See “Detecting authorization failures (EXTENDED SECURITY)” on page 177 for information about security diagnostics.

1. You must first decide whether you want incoming requests to have authentication information, such as RACF passwords, RACF PassTickets, and Kerberos tickets, passed along with the authorization ID.

   To indicate that you require this authentication information, specify NO on the TCP/IP ALREADY VERIFIED field of installation panel DSNTIP5, which is the default option. If you do not specify NO, all incoming TCP/IP requests can connect to DB2 without any authentication.

2. If you require authentication, ensure that the security subsystem at your server is properly configured to handle the authentication information that is passed to it.
   - For requests that use RACF passwords or PassTickets, enter the following RACF command to indicate which user IDs that use TCP/IP are authorized to access DDF (the distributed data facility address space):

```
PERMIT ssnm.DIST CLASS(DSNR) ID(yyy) ACCESS(READ)
  WHEN(APPCPORT(TCPIP))
```

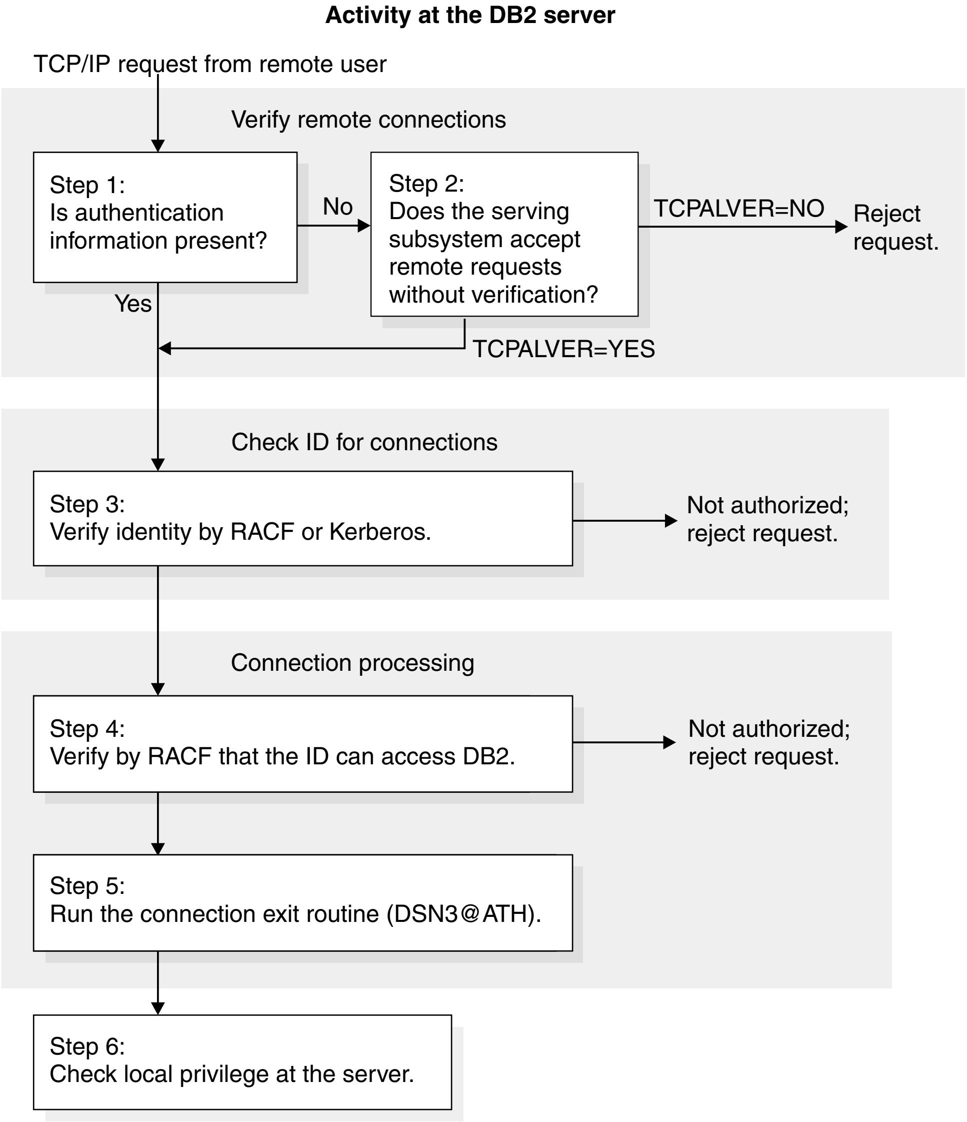

Activity at the DB2 server
TCP/IP request from remote user
Verify remote connections
Step 1:
Is authentication information present?
No
Step 2:
Does the serving subsystem accept remote requests without verification?
TCPALVER=NO
Reject request.
Yes
TCPALVER=YES
Check ID for connections
Step 3:
Verify identity by RACF or Kerberos.
Not authorized; reject request.
Connection processing
Step 4:
Verify by RACF that the ID can access DB2.
Not authorized; reject request.
Step 5:
Run the connection exit routine (DSN3@ATH).
Step 6:
Check local privilege at the server.

*Figure 18. Steps in accepting a request from TCP/IP.*

***Details of steps:*** These notes explain the steps shown in Figure 18.

1. DB2 checks to see if an authentication token (RACF encrypted password, RACF PassTicket, DRDA encrypted password, or Kerberos ticket) accompanies the remote request.

2. If no authentication token is supplied, DB2 checks the TCPALVER subsystem parameter to see if DB2 accepts IDs without authentication information. If TCPALVER=NO, authentication information must accompany all requests, and DB2 rejects the request. If TCPALVER=YES, DB2 accepts the request without authentication.
3. The identity is a RACF ID that is authenticated by RACF if a password or PassTicket is provided, or the identity is a Kerberos principal that is validated by Kerberos Security Server, if a Kerberos ticket is provided. Ensure that the ID is defined to RACF in all cases. When Kerberos tickets are used, the RACF ID is derived from the Kerberos principal identity. To use Kerberos tickets, ensure that you map Kerberos principal names with RACF IDs, as described in "Establishing Kerberos authentication through RACF" on page 212.

   In addition, depending on your RACF environment, the following RACF checks may also be performed:

   a. If the RACF APPL class is active, RACF verifies that the ID has access to the DB2 APPL. The APPL resource that is checked is the LU name that the requester used when the attachment request was issued. This is either the local DB2 LU name or the generic LU name.
   b. If the RACF APPCPORT class is active, RACF verifies that the ID is authorized access to MVS from the port of entry (POE). The POE that is used in the verify call is the string 'TCPIP'.

   If this is a request to change a password, the password is changed.
4. The remote request is now treated like a local connection request (using the DIST environment for the DSNR resource class). DB2 calls RACF to check the ID's authorization against the *ssnm*.DIST resource.
5. DB2 invokes the connection exit routine. The parameter list that is passed to the routine describes where the remote request originated.
6. The remote request has a primary authorization ID, possibly one or more secondary IDs, and an SQL ID. (The SQL ID cannot be translated.) The plan or package owner ID also accompanies the request. Privileges and authorities that are granted to those IDs at the DB2 server govern the actions that the request can take.

## Planning to send remote requests

If you are planning to send requests to another DB2 subsystem, consider that the security administrator of that subsystem might have chosen any of the options described in "Controlling requests from remote applications" on page 176. You need to know what those choices are and make entries in your CDB to correspond to them. You can also choose some things independently of what the other subsystem requires.

If you are planning to send remote requests to a DBMS that is not DB2 for OS/390 and z/OS, you need to satisfy the requirements of that system. You probably need documentation for the particular type of system; some of the choices that are described in this section might not apply.

***Network protocols and authentication tokens:*** DB2 chooses how to send authentication tokens based on the network protocols that are used (SNA or TCP/IP). If the request is sent using SNA, the authentication tokens are sent in the SNA attachment request (FMH5), unless you are using Kerberos. If you use Kerberos, authentication tokens are sent with DRDA security commands.

If the request uses TCP/IP, the authentication tokens are always sent using DRDA security commands.

## The communications database for the requester

The information in this section, up to “What IDs you send” on page 193, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

The communications database (CDB) is a set of DB2 catalog tables that let you control aspects of remote requests. This section concentrates on the columns of the communications database that pertain to security issues related to the requesting system.

### Columns used in SYSIBM.LUNAMES

This table is used only for requests that use SNA protocols.

**LUNAME CHAR(8)**

The LUNAME of the remote system. A blank value identifies a default row that serves requests from any system that is not specifically listed elsewhere in the column.

**SECURITY_OUT (CHAR 1)**

Indicates the security option that is used when local DB2 SQL applications connect to any remote server that is associated with the corresponding LUNAME.

**A** The option is “already verified”, the default. With A, outbound connection requests contain an authorization ID and no authentication token. The value that is used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the value in the USERNAMES column.

**R** The option is “RACF PassTicket”. Outbound connection requests contain a user ID and a RACF PassTicket. The LUNAME column is used as the RACF PassTicket application name.

The value that is used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the value in the USERNAMES column. The translated ID is used to build the RACF PassTicket. Do not specify R for CONNECT statements with a USER parameter.

**P** The option is “password”. Outbound connection requests contain an authorization ID and a password. The password is obtained from RACF if ENCRYPTPSWDS=Y, or from SYSIBM.USERNAMES if ENCRYPTPSWDS=N. If you get the password from SYSIBM.USERNAMES, the USERNAMES column of SYSIBM.LUNAMES must contain B or O. The value that is used for an outbound request is the translated ID.

**ENCRYPTPSWDS CHAR(1)**

Indicates whether passwords received from and sent to the corresponding LUNAME are encrypted. This column only applies to DB2 for OS/390 and z/OS and DB2 for MVS/ESA partners when passwords are used as authentication tokens.

**Y** Yes, passwords are encrypted. For outbound requests, the encrypted password is extracted from RACF and sent to the server. For inbound requests, the password is treated as encrypted.

**N** No, passwords are not encrypted. This is the default; any character but Y is treated as N.

**Recommendation:** When you connect to a DB2 for OS/390 and z/OS partner that is at Version 5 or a subsequent release, use RACF PassTickets (SECURITY_OUT='R') instead of encrypting passwords.

**USERNAMES CHAR(1)**

Indicates whether an ID accompanying a remote attachment request, which is received from or sent to the corresponding LUNAME, is subject to translation and “come from” checking. When you specify I, O, or B, use the SYSIBM.USERNAMES table to perform the translation.

**I** An inbound ID is subject to translation.

**O** An outbound ID, sent to the corresponding LUNAME, is subject to translation.

**B** Both inbound and outbound IDs are subject to translation.

**blank** No IDs are translated.

## Columns used in SYSIBM.IPNAMES

This table is used only for requests that use TCP/IP protocols.

**LINKNAME CHAR(8)**

The name used in the LINKNAME column of SYSIBM.LOCATIONS to identify the remote system.

**SECURITY_OUT**

Indicates the DRDA security option that is used when local DB2 SQL applications connect to any remote server that is associated with this TCP/IP host.

**A** The option is “already verified”, the default. Outbound connection requests contain an authorization ID and no password. The value that is used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the value in the USERNAMES column.

**R** The option is “RACF PassTicket”. Outbound connection requests contain a user ID and a RACF PassTicket. The LINKNAME column must contain the server's LU name, which is used as the RACF PassTicket application name to generate the PassTicket.

The value that is used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the value in the USERNAMES column. The translated ID is used to build the RACF PassTicket. Do not specify R for CONNECT statements with a USER parameter.

**P** The option is “password”. Outbound connection requests contain an authorization ID and a password. The password is obtained from the SYSIBM.USERNAMES table.

If you specify P, the USERNAMES column must contain O.

**USERNAMES CHAR(1)**

This column indicates whether an outbound request translates the authorization ID. When you specify O, use the SYSIBM.USERNAMES table to perform the translation.

**O** An outbound ID, sent to the corresponding LUNAME, is subject to translation.

**blank** No translation is done.

## Columns used in SYSIBM.USERNAMES

This table is used by both SNA and TCP/IP connections.

**TYPE CHAR(1)**

Indicates whether the row is used for inbound or outbound translation:

**I** The row applies to inbound IDs.

**O** The row applies to outbound IDs.

The field should contain only I or O. Any other character, including blank, causes the row to be ignored.

**AUTHID CHAR(8)**

An authorization ID that is permitted and perhaps translated. If blank, any authorization ID is permitted with the corresponding LINKNAME, and all authorization IDs are translated in the same way.

**LINKNAME CHAR(8)**

Identifies the VTAM or TCP/IP network locations that are associated with this row. A blank value in this column indicates that this name translation rule applies to any TCP/IP or SNA partner.

If you specify a nonblank value for this column, one or both of the following situations must be true:

- A row exists in table SYSIBM.LUNAMES that has an LUNAME value that matches the LINKNAME value that appears in this column.
- A row exists in table SYSIBM.IPNAMES that has a LINKNAME value that matches the LINKNAME value that appears in this column.

**NEWAUTHID CHAR(8)**

The translated authorization ID. If blank, no translation occurs.

**PASSWORD CHAR(8)**

A password that is sent with outbound requests. This password is not provided by RACF and cannot be encrypted.

## Columns used in SYSIBM.LOCATIONS

This table controls which locations can access DB2. If you use DB2 private protocol access, the remote location name must be specified in this table. This check is only supported for connections using private protocols.

**LOCATION CHAR(16)**

**LINKNAME CHAR(8)**

Identifies the VTAM or TCP/IP network locations that are associated with this row. A blank value in this column indicates that this name translation rule applies to any TCP/IP or SNA partner.

If you specify a nonblank value for this column, one or both of the following situations must be true:

- A row exists in table SYSIBM.LUNAMES that has an LUNAME value that matches the LINKNAME value that appears in this column.
- A row exists in table SYSIBM.IPNAMES that has a LINKNAME value that matches the LINKNAME value that appears in this column.

**PORT CHAR(32)**

TCP/IP is used for outbound DRDA connections when the following statement is true:

- A row exists in SYSIBM.IPNAMES, where the LINKNAME column matches the value specified in the SYSIBM.LOCATIONS LINKNAME column.

If the above-mentioned row is found, the value of the PORT column is interpreted as follows:

- If PORT is blank, the default DRDA port (446)is used.
- If PORT is nonblank, the value specified for PORT can take one of two forms:
  - If the value in PORT is left justified with 1-5 numeric characters, the value is assumed to be the TCP/IP port number of the remote database server.
  - Any other value is assumed to be a TCP/IP service name, which can be converted to a TCP/IP port number using the TCP/IP getservbyname socket all. TCP/IP service names are not case-sensitive.

**TPN VARCHAR(64)**

Used only when the local DB2 begins an SNA conversation with another server. When used, TPN indicates the SNA LU 6.2 transaction program name (TPN) that will allocate the conversation. A length of zero for the column indicates the default TPN. For DRDA conversations, this is the DRDA default, which is X'07F6C4C2'.

For DB2 private protocol conversations, this column is not used. For an SQL/DS server, TPN should contain the resource ID of the SQL/DS machine.

## What IDs you send

At least one authorization ID is always sent to the server to be used for authentication. That ID is one of the following values:

- The primary authorization ID of the process.
- If you connect to the server using a CONNECT statement with the USER keyword, the ID that you specify as the USER ID.

However, other IDs can accompany some requests. You need to understand what other IDs are sent, because they are subject to translation. You must include these other IDs in table SYSIBM.USERNAMES to avoid an error when you use outbound translation. Table 54 shows what other IDs you send for the different situations that can occur.

*Table 54. IDs that accompany the primary ID on a remote request*

| In this situation: | You send this ID also: |
|---|---|
| An SQL query, using DB2 private-protocol access | The plan owner. |
| A remote BIND, COPY, or REBIND PACKAGE command | The package owner. |

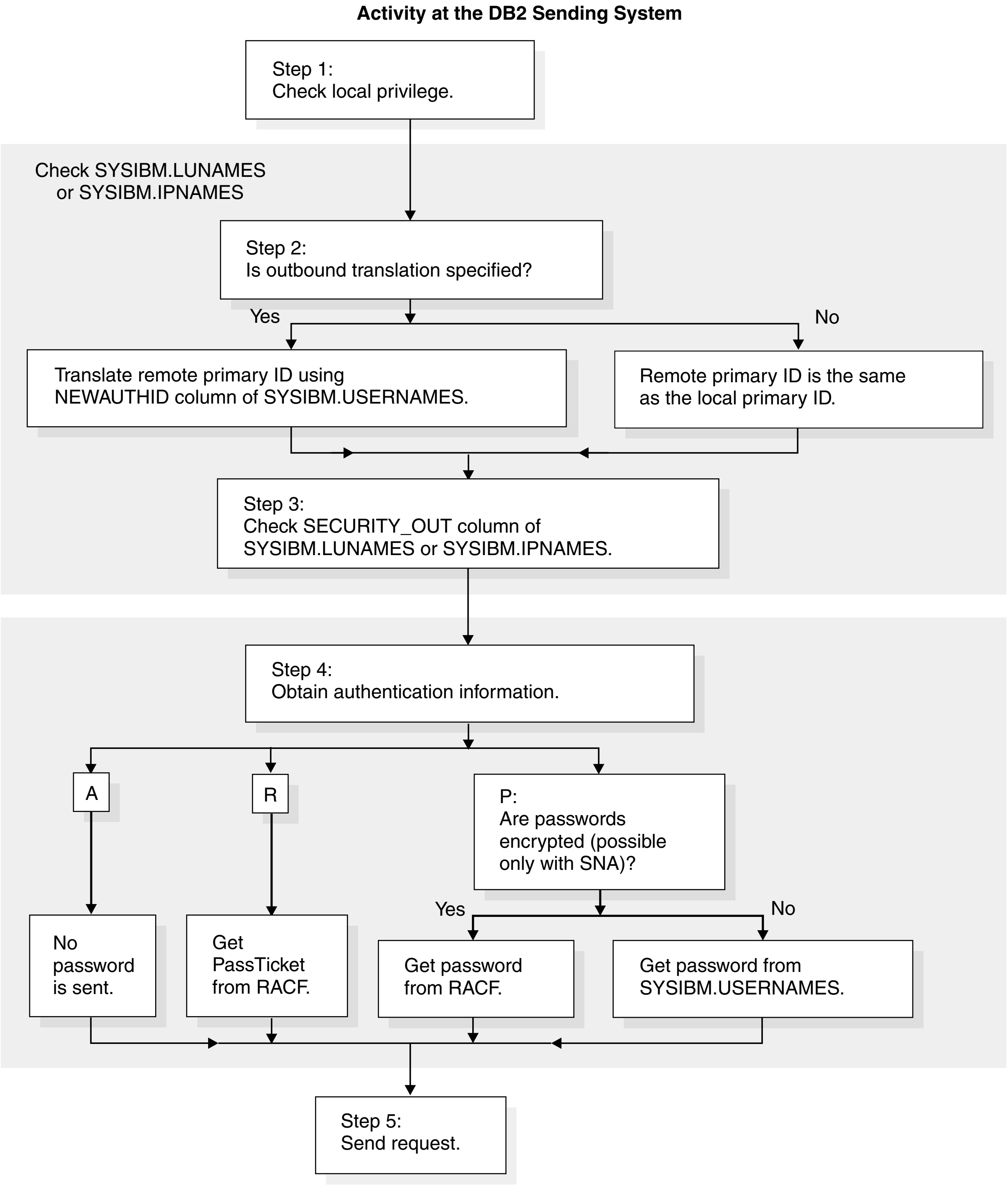


*Figure 19. Steps in sending a request from a DB2 subsystem*

***Details of steps in sending a request from DB2:*** These notes explain the steps in Figure 19.

1. The DB2 subsystem that sends the request checks whether the primary authorization ID has the privilege to execute the plan or package.

   DB2 determines what value in column LINKNAME of table SYSIBM.LOCATIONS matches either column LUNAME of table

SYSIBM.LUNAMES or column LINKNAME of table SYSIBM.IPNAMES. This check determines whether SNA or TCP/IP protocols are used to carry the DRDA request. (Statements that use DB2 private protocol, not DRDA, always use SNA.)

2. When executing a plan, the plan owner is also sent with the authorization ID; when binding a package, the authorization ID of the package owner is also sent. If the USERNAMES column of table SYSIBM.LUNAMES contains O or B, or if the USERNAMES column of table SYSIBM.IPNAMES contains O, both IDs are subject to translation under control of the SYSIBM.USERNAMES table. Ensure that these IDs are included in SYSIBM.USERNAMES, or SQLCODE -904 is issued. DB2 translates the ID as follows:
   - If a nonblank value of NEWAUTHID is in the row, that value becomes the new ID.
   - If NEWAUTHID is blank, the ID is not changed.

   If table SYSIBM.USERNAMES does not contain a new authorization ID to which the primary authorization ID is translated, the request is rejected with a SQLCODE -904.

   If column USERNAMES does not contain O or B, the IDs are not translated.
3. SECURITY_OUT is checked for outbound security options, as follows:

   **A** Already verified. No password is sent with the authorization ID. This option is valid only if the server accepts already verified requests.
   For SNA, the server must have specified A in the SECURITY_IN column of the SYSIBM.LUNAMES table.
   For TCP/IP, the server must have specified YES in the TCP/IP ALREADY VERIFIED field of installation panel DSNTIP5.

   **R** RACF PassTicket. If the primary authorization ID was translated, that translated ID is sent with the PassTicket. See “Sending RACF PassTickets” on page 197 for information about setting up PassTickets.

   **P** Password. The outbound request must be accompanied by a password:
   If the requester is a DB2 for OS/390 and z/OS and uses SNA protocols, passwords can be encrypted if you specify Y in the ENCRYPTPSWDS column of SYSIBM.LUNAMES. If passwords are not encrypted, the password is obtained from the PASSWORD column of table SYSIBM.USERNAMES.

   **Recommendation:** Use RACF PassTickets to avoid flowing unencrypted passwords over the network.
   If the requester uses TCP/IP protocols, you cannot encrypt the password; therefore, the password is always obtained from RACF.
4. Send the request. See Table 54 on page 193 to determine which IDs accompany the primary authorization ID.

## Translating outbound IDs

One reason for translating outbound IDs is that an ID on your system duplicates an ID on the remote system. Or, you might want to change some IDs to others that *are* accepted by the remote system.

To indicate that you want to translate outbound user IDs:

1. Specify an O in the USERNAMES column of table SYSIBM.IPNAMES or SYSIBM.LUNAMES.

2. Use the NEWAUTHID column of SYSIBM.USERNAMES to specify the ID to which the outbound ID is translated.

***Example 1:*** Suppose that the remote system accepts from you only the IDs XXGALE, GROUP1, and HOMER.

1. To specify that outbound translation is in effect for the remote system, LUXXX, you need the following values in table SYSIBM.LUNAMES:

| LUNAME | USERNAMES |
|---|---|
| LUXXX | O |

   If your row for LUXXX already has I for column USERNAMES (because you translate inbound IDs that come from LUXXX), change I to B (for both inbound and outbound translation.

2. Translate the ID GALE to XXGALE on all outbound requests to LUXXX. You need these values in table SYSIBM.USERNAMES:

| TYPE | AUTHID | LINKNAME | NEWAUTHID | PASSWORD |
|---|---|---|---|---|
| O | GALE | LUXXX | XXGALE | GALEPASS |

3. Translate EVAN and FRED to GROUP1 on all outbound requests to LUXXX. You need these values in table SYSIBM.USERNAMES:

| TYPE | AUTHID | LINKNAME | NEWAUTHID | PASSWORD |
|---|---|---|---|---|
| O | EVAN | LUXXX | GROUP1 | GRP1PASS |
| O | FRED | LUXXX | GROUP1 | GRP1PASS |

4. Do not translate the ID HOMER on outbound requests to LUXXX. (HOMER is assumed to be an ID on your DB2, and on LUXXX.) You need these values in table SYSIBM.USERNAMES:

| TYPE | AUTHID | LINKNAME | NEWAUTHID | PASSWORD |
|---|---|---|---|---|
| O | HOMER | LUXXX | blank | HOMERSPW |

5. Reject any requests from BASIL to LUXXX before they are sent. For that, you need nothing in table SYSIBM.USERNAMES. If no row indicates what to do with the ID BASIL on an outbound request to LUXXX, the request is rejected.

***Example 2:*** If you send requests to another LU, such as LUYYY, you generally need another set of rows to indicate how your IDs are to be translated on outbound requests to LUYYY.

However, you can use a single row to specify a translation that is to be in effect on requests to all other LUs. For example, if HOMER is to be sent untranslated everywhere, and DOROTHY is to be translated to GROUP1 everywhere, you can use these values in table SYSIBM.USERNAMES:

| TYPE | AUTHID | LINKNAME | NEWAUTHID | PASSWORD |
|---|---|---|---|---|
| O | HOMER | blank | blank | HOMERSPW |
| O | DOROTHY | blank | GROUP1 | GRP1PASS |

You can also use a single row to specify that all IDs that accompany requests to a single remote system must be translated. For example, if every one of your IDs is to be translated to THEIRS on requests to LUYYY, you can use the following values in table SYSIBM.USERNAMES:

| TYPE | AUTHID | LINKNAME | NEWAUTHID | PASSWORD |
|---|---|---|---|---|
| O | blank | LUYYY | THEIRS | THEPASS |

## Sending passwords

**Recommendation:** For the tightest security, do not send passwords through the network. Instead, use one of the following security mechanisms:

- RACF encrypted passwords, described in “Sending RACF encrypted passwords”
- RACF PassTickets, described in “Sending RACF PassTickets”
- Kerberos tickets, described in “Establishing Kerberos authentication through RACF” on page 212
- DRDA encrypted passwords or DRDA encrypted user IDs with encrypted passwords, described in “Sending encrypted passwords from a workstation” on page 198

If you want to send passwords, you can put the password for an ID in the PASSWORD column of SYSIBM.USERNAMES. **If you do this, pay special attention to the security of the SYSIBM.USERNAMES table.** We strongly recommend that you use an edit routine (EDITPROC) to encrypt the passwords and authorization IDs in SYSIBM.USERNAMES. For instructions on writing an edit routine and creating a table that uses it, see “Edit routines” on page 921.

DB2 for OS/390 and z/OS allows the use of RACF encrypted passwords or RACF PassTickets. However, workstations, such as Windows NT®, do not support these security mechanisms. RACF encrypted passwords are not a secure mechanism, because they can be replayed.

**Recommendation**: Do not use RACF encrypted passwords unless you are connecting to a previous release of DB2 for OS/390 and z/OS.

### Sending RACF encrypted passwords

A method available only to DB2 subsystems that communicate with each other using SNA protocols is to specify password encryption in SYSIBM.LUNAMES as follows:

| LUNAME | USERNAMES | ENCRYPTPSWDS |
|---|---|---|
| LUXXX | O | Y |

The partner DB2 must also specify password encryption in its SYSIBM.LUNAMES table. Both partners must register each ID and its password with RACF. Then, for every request to LUXXX, your DB2 calls RACF to supply an encrypted password to accompany the ID. With password encryption, you do not use the PASSWORD column of SYSIBM.USERNAMES, so the security of that table becomes less critical.

### Sending RACF PassTickets

To send RACF PassTickets with your remote requests to a particular remote system, enter R in the SECURITY_OUT column of the SYSIBM.IPNAMES or SYSIBM.LUNAMES table for that system.

To set up RACF to generate PassTickets, define and activate a RACF PassTicket data class (PTKTDATA). This class must contain a RACF profile for each remote DB2 subsystem to which you send requests.

1. Activate the RACF PTKTDATA class by issuing the following RACF commands:

```
SETROPTS CLASSACT(PTKTDATA)
SETROPTS RACLIST(PTKTDATA)
```

2. Define profiles for the remote systems by entering the name of each remote system as it appears in the LINKNAME column of table SYSIBM.LOCATIONS. For example, the following command defines a profile for a remote system, DB2A, in the RACF PTKTDATA class:

   ```
   RDEFINE PTKTDATA DB2A SSIGNON(KEYMASKED(E001193519561977))
   ```

3. Refresh the RACF PTKTDATA definition with the new profile by issuing the following command:

   ```
   SETROPTS RACLIST(PTKTDATA) REFRESH
   ```

See *OS/390 Security Server (RACF) Security Administrator's Guide* for more information about RACF PassTickets.

### Sending encrypted passwords from a workstation

Depending on the DRDA level, clients can encrypt passwords or user IDs and passwords when they send them to a DB2 for OS/390 and z/OS server. This support uses the Diffie-Hellman key distribution algorithm.[6]

To enable DB2 Connect to flow encrypted passwords, database connection services (DCS) authentication must be set to DCS_ENCRYPT in the DCS directory entry. When the workstation application issues an SQL CONNECT, the workstation negotiates this support with the database server. If supported, a shared private key is generated by the client and server using the Diffie-Hellman public key technology and the password is encrypted using 56-bit DES with the shared private key. The encrypted password is non-replayable, and the shared private key is generated on every connection. If the server does not support password encryption, the application receives SQLCODE -30073 (DRDA security manager level 6 is not supported).

## Establishing RACF protection for DB2

For purposes of illustration, suppose that the system of RACF IDs that is shown in Figure 20 on page 199 is used to control DB2 usage.

[6] Diffie-Hellman is one of the first standard public key algorithms. It results in exchanging a connection key which is used by client and server to generate a shared private key. The 56-bit Data Encryption Standards (DES) algorithm is used for encrypting and decrypting of the password using the shared private key.

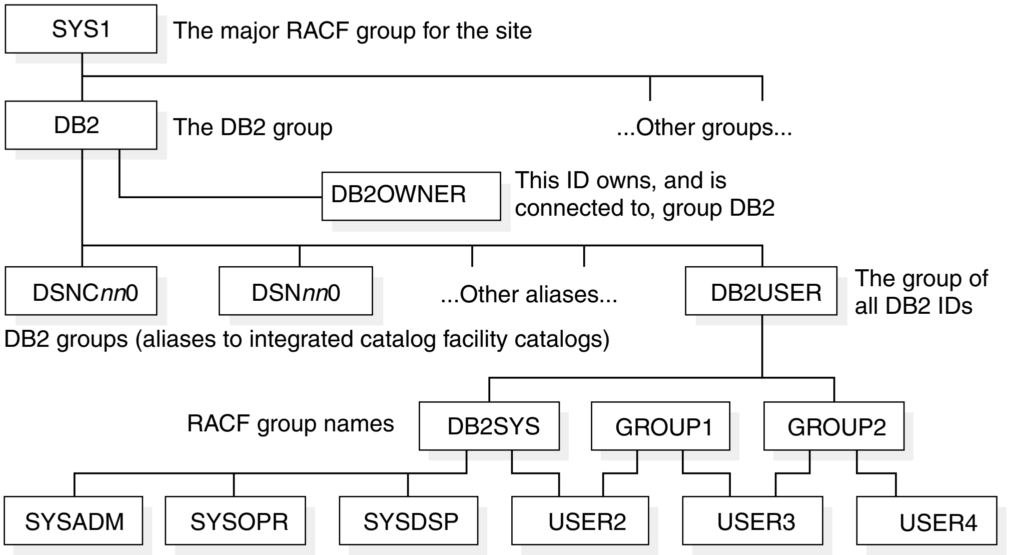


*Figure 20. Sample DB2–RACF environment*

Figure 20 shows some of the relationships among the following names:

*Table 55. RACF Relationships*

| RACF ID | Use |
|---|---|
| SYS1 | Major RACF group ID |
| DB2 | DB2 group |
| DB2OWNER | Owner of the DB2 group |
| DSNC710 | Group to control databases and recovery logs |
| DSN710 | Group to control installation data sets |
| DB2USER | Group of all DB2 users |
| SYSADM | ID with DB2 installation SYSADM authority |
| SYSOPR | ID with DB2 installation SYSOPR authority |
| DB2SYS, GROUP1, GROUP2 | RACF group names |
| SYSDSP | RACF user ID for DB2 started tasks |
| USER1, USER2, USER3<br>**Note:** These RACF group names and user IDs do not appear in the figure; they are listed in Table 56 on page 204. | RACF user IDs |

To establish RACF protection for DB2, perform the steps described in the following two sections. Some are required and some are optional, depending on your circumstances. All presume that RACF is already installed. The steps do not need to be taken strictly in the order shown here; they are grouped under two major objectives:

- “Defining DB2 resources to RACF” on page 200 includes steps that tell RACF what to protect.
- “Permitting RACF access” on page 202 includes steps that make the protected resources available to processes.

For a more thorough description of RACF facilities, see *OS/390 Security Server (RACF) System Programmer's Guide*.

## Defining DB2 resources to RACF

To define DB2 resources to the RACF system, perform the following required steps in any order:

- "Define the names of protected access profiles"
- "Add entries to the RACF router table" on page 201
- "Enable RACF checking for the DSNR and SERVER classes" on page 202

No one can access the DB2 subsystem until you instruct RACF to permit access.

Other tasks you might want to perform include:

- Controlling whether two DBMSs using VTAM LU 6.2 can establish sessions with each other, as described in "Enable partner-LU verification" on page 202.
- Ensure that IDs that are associated with stored procedures address spaces are authorized to run the appropriate attachment facility, as described in "Step 1: Control access by using the attachment facilities (required)" on page 209.
- If you are using TCP/IP, ensure that the ID that is associated with the DDF address space is authorized to use OS/390 UNIX (formerly called OpenEdition®n) services, as described in "Establishing RACF protection for TCP/IP" on page 212.

### Define the names of protected access profiles

The RACF resource class for DB2 is DSNR, and that class is contained in the RACF descriptor table. Among the resources in that class are profiles for access to a DB2 subsystem from one of these environments—IMS, CICS, the distributed data facility (DDF), TSO, CAF, or batch. Those profiles allow you to control access to a DB2 subsystem from a particular environment.

Each profile has a name of the form *subsystem.environment*, where:

- *subsystem* is the name of a DB2 subsystem, of one to four characters; for example, DSN or DB2T.
- *environment* denotes the environment, by one of the following terms:
  - MASS for IMS (including MPP, BMP, Fast Path, and DL/I batch).
  - SASS for CICS.
  - DIST for DDF.
  - RRSAF for Recoverable Resource Manager Services attachment facility. Stored procedures use RRSAF in WLM-established address spaces.
  - BATCH for all others, including TSO, CAF, batch, all utility jobs, DB2-established stored procedures address space, and requests that come through the call attachment facility.

To control access, you need to define a profile name, as a member of class DSNR, for every combination of subsystem and environment you want to use. For example, suppose you want to access:

- Subsystem DSN from TSO and DDF
- Subsystem DB2P from TSO, DDF, IMS, and RRSAF
- Subsystem DB2T from TSO, DDF, CICS, and RRSAF

Then define the following profile names:

```
DSN.BATCH   DSN.DIST
DB2P.BATCH  DB2P.DIST  DB2P.MASS  DB2P.RRSAF
DB2T.BATCH  DB2T.DIST  DB2T.SASS  DB2T.RRSAF
```

You can do that with a single RACF command, which also names an owner for the resources:

```
RDEFINE DSNR (DSN.BATCH  DSN.DIST  DB2P.BATCH  DB2P.DIST  DB2P.MASS DB2P.RRSAF
              DB2T.BATCH DB2T.DIST DB2T.SASS DB2T.RRSAF)  OWNER(DB2OWNER)
```

Those profiles are the ones that you later permit access to, as shown under “Permit access for users and groups” on page 207. After you define an entry for your DB2 subsystem in the RACF router table, the only users that can access the system are those who are permitted access to a profile. If you do not want to limit access to particular users or groups, you can give universal access to a profile with a command like this:

```
RDEFINE DSNR (DSN.BATCH) OWNER(DB2OWNER) UACC(READ)
```

When you have added an entry for an DB2 subsystem to the RACF router table, you must remove the entry for that subsystem from the router table to deactivate RACF checking.

### Add entries to the RACF router table

You need to add an entry for each DB2 subsystem to the RACF router table, because they are not included in the default router table that is distributed by RACF. Figure 21 on page 202 shows the ICHRFRTB macros to generate entries in the RACF router table (ICHRFR01) for the DB2 subsystems DSN, DB2P, and DB2T. This table controls the action that is taken when DB2 invokes the RACROUTE macro. (Refer to *OS/390 Security Server (RACF) System Programmer's Guide* for a description of how to generate the RACF router table and the RACROUTE macro). If you do not have an entry in the router table for a particular DB2 subsystem, any user who tries to access that subsystem from any environment is accepted.

If you later decide not to use RACF checking for any or all of these resources, use the RACF RDELETE command to delete the resources you do not want checked. Then reassemble the RACF router table without them.

Finally, perform an IPL of the MVS system to cause it to use the new router table. Alternatively, you can delay the IPL until you have reassembled the RACF started procedures table in the next set of steps and, therefore, do it only once.

***Tip:*** The macro ICHRFRTB that is used in the job sends a message to warn that the class name DSNR does not contain a digit or national character in the first four characters. You can ignore the message.

As a result of the job, RACF (ACTION=RACF) receives the requests of those subsystems, in class DSNR with requester IDENTIFY, for connection checking.

```
*
*  REASSEMBLE AND LINKEDIT THE INSTALLATION-PROVIDED
*  ROUTER TABLE ICHRFR01 TO INCLUDE DB2 SUBSYSTEMS IN THE
*  DSNR RESOURCE CLASS.
*
*  PROVIDE ONE ROUTER ENTRY FOR EACH DB2 SUBSYSTEM NAME.
*  THE REQUESTOR-NAME MUST ALWAYS BE "IDENTIFY".

ICHRFRTB CLASS=DSNR,REQSTOR=IDENTIFY,SUBSYS=DSN,ACTION=RACF
ICHRFRTB CLASS=DSNR,REQSTOR=IDENTIFY,SUBSYS=DB2P,ACTION=RACF
ICHRFRTB CLASS=DSNR,REQSTOR=IDENTIFY,SUBSYS=DB2T,ACTION=RACF
```

*Figure 21. Router table in RACF*

### Enable RACF checking for the DSNR and SERVER classes

To enable RACF access checking for resources in the DSNR resource class, issue this RACF command:

```
SETROPTS CLASSACT(DSNR)
```

Only users with the SPECIAL attribute can issue the command.

If you are using stored procedures in a WLM-established address space, you might also need to enable RACF checking for the SERVER class. See “Step 2: Control access to WLM (optional)” on page 210.

### Enable partner-LU verification

With RACF 1.9, VTAM 3.3, and later releases of either product, you can control whether two LUs that use LU 6.2 can connect to each other.

Each member of a connecting pair must establish a profile for the other member. For example, if LUAAA and LUBBB are to connect and know each other by those LUNAMES, issue RACF commands similar to these:

At LUAAA: `RDEFINE  APPCLU  ` *netid*`.LUAAA.LUBBB UACC(NONE)` ...
At LUBBB: `RDEFINE  APPCLU  ` *netid*`.LUBBB.LUAAA UACC(NONE)` ...

Here, *netid* is the network ID, given by the VTAM start option NETID.

When you create those profiles with RACF, use the SESSION operand to supply:
- The VTAM password as a session key (SESSKEY suboperand)
- The maximum number of days between changes of the session key (INTERVAL suboperand)
- An indication of whether the LU pair is locked (LOCK suboperand)

For details, see *OS/390 Security Server (RACF) Security Administrator's Guide*.

Finally, to enable RACF checking for the new APPCLU resources, issue this RACF command at both LUAAA and LUBBB:

```
SETROPTS CLASSACT(APPCLU)
```

## Permitting RACF access

To let processes use the protected resources, take the steps described in the following sections:
1. “Define RACF user IDs for DB2 started tasks” on page 203
2. “Add RACF groups” on page 206
3. “Permit access for users and groups” on page 207

The sections that follow provide detailed suggestions.

## Define RACF user IDs for DB2 started tasks

A DB2 subsystem has the following started-task address spaces:

- *ssnm*DBM1 for database services
- *ssnm*MSTR for system services
- *ssnm*DIST for the distributed data facility
- *ssnm*SPAS for the DB2-established stored procedures address space
- Names for your WLM-established address spaces for stored procedures

You must associate each of these address spaces with a RACF user ID. Each of them can also be assigned a RACF group name. The DB2 address spaces *cannot* be started with batch jobs.

If you have IMS or CICS applications issuing DB2 SQL requests, you must associate RACF user IDs, and can associate group names, with:

- The IMS control region
- The CICS address space
- The four DB2 address spaces

If the IMS and CICS address spaces are started as batch jobs, provide their RACF IDs and group names with the USER and GROUP parameters on the JOB statement. If they are started as started tasks, assign the IDs and group names as you do for the DB2 address spaces, by changing the RACF started procedures table.

| | | | | | | ***Stored procedures:*** Entries for stored procedures address spaces are required in the RACF started procedures table. The associated RACF user ID and group name do not need to match those that are used for the DB2 address spaces, but they must be authorized to run the call attachment facility (for the DB2-established stored procedures address space) or Recoverable Resource Manager Services attachment facility (for WLM-established stored procedures address spaces). Note: WLM-established stored procedures started tasks IDs require an OMVS segment.

***Changing the RACF started-procedures table:*** To change the RACF started-procedures table (ICHRIN03), change, reassemble, and link edit the resulting object code to MVS. Figure 22 on page 204 shows the sample entries for three DB2 subsystems and optional entries for CICS and IMS. (Refer to *OS/390 Security Server (RACF) System Programmer's Guide* for a description of how to code a RACF started-procedures table.) The example provides the DB2 started tasks for each of three DB2 subsystems, named DSN, DB2T, and DB2P, and for CICS and an IMS control region.

The IDs and group names associated with the address spaces are shown in Table 56 on page 204.

*Table 56. DB2 address space IDs and associated RACF user IDs and group names*

| Address Space | RACF User ID | RACF Group Name |
|---|---|---|
| DSNMSTR | SYSDSP | DB2SYS |
| DSNDBM1 | SYSDSP | DB2SYS |
| DSNDIST | SYSDSP | DB2SYS |
| DSNSPAS | SYSDSP | DB2SYS |
| DSNWLM | SYSDSP | DB2SYS |
| DB2TMSTR | SYSDSPT | DB2TEST |
| DB2TDBM1 | SYSDSPT | DB2TEST |
| DB2TDIST | SYSDSPT | DB2TEST |
| DB2TSPAS | SYSDSPT | DB2TEST |
| DB2PMSTR | SYSDSPD | DB2PROD |
| DB2PDBM1 | SYSDSPD | DB2PROD |
| DB2PDIST | SYSDSPD | DB2PROD |
| DB2PSPAS | SYSDSPD | DB2PROD |
| CICSSYS | CICS | CICSGRP |
| IMSCNTL | IMS | IMSGRP |

```
//*
//*  REASSEMBLE AND LINKEDIT THE RACF STARTED-PROCEDURES
//*  TABLE ICHRIN03 TO INCLUDE USERIDS AND GROUP NAMES
//*  FOR EACH DB2 CATALOGED PROCEDURE. OPTIONALLY, ENTRIES
//*  FOR AN IMS OR CICS SYSTEM MIGHT BE INCLUDED.
//*
//*  AN IPL WITH A CLPA (OR AN MLPA SPECIFYING THE LOAD
//*  MODULE) IS REQUIRED FOR THESE CHANGES TO TAKE EFFECT.
//*

ENTCOUNT DC    AL2(((ENDTABLE-BEGTABLE)/ENTLNGTH)+32768)
*              NUMBER OF ENTRIES AND INDICATE RACF FORMAT
*
*  PROVIDE FOUR ENTRIES FOR EACH DB2 SUBSYSTEM NAME.
*
```

*Figure 22. Sample job to reassemble the RACF started-procedures table (Part 1 of 5)*

```
BEGTABLE DS    0H
*        ENTRIES FOR SUBSYSTEM NAME "DSN"
         DC    CL8'DSNMSTR'        SYSTEM SERVICES PROCEDURE
         DC    CL8'SYSDSP'         USERID
         DC    CL8'DB2SYS'         GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
ENTLNGTH EQU   *-BEGTABLE     CALCULATE LENGTH OF EACH ENTRY
         DC    CL8'DSNDBM1'   DATABASE SERVICES PROCEDURE
         DC    CL8'SYSDSP'         USERID
         DC    CL8'DB2SYS'         GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DSNDIST'        DDF PROCEDURE
         DC    CL8'SYSDSP'         USERID
         DC    CL8'DB2SYS'         GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DSNSPAS'        STORED PROCEDURES PROCEDURE
         DC    CL8'SYSDSP'         USERID
         DC    CL8'DB2SYS'         GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DSNWLM'         WLM-ESTABLISHED S.P. ADDRESS SPACE
         DC    CL8'SYSDSP'         USERID
         DC    CL8'DB2SYS'         GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
```

*Figure 22. Sample job to reassemble the RACF started-procedures table (Part 2 of 5)*

```
*        ENTRIES FOR SUBSYSTEM NAME "DB2T"
         DC    CL8'DB2TMSTR'       SYSTEM SERVICES PROCEDURE
         DC    CL8'SYSDSPT'        USERID
         DC    CL8'DB2TEST'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DB2TDBM1'     DATABASE SERVICES PROCEDURE
         DC    CL8'SYSDSPT'        USERID
         DC    CL8'DB2TEST'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DB2TDIST'       DDF PROCEDURE
         DC    CL8'SYSDSPT'        USERID
         DC    CL8'DB2TEST'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DB2TSPAS'       STORED PROCEDURES PROCEDURE
         DC    CL8'SYSDSPT'        USERID
         DC    CL8'DB2TEST'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
```

*Figure 22. Sample job to reassemble the RACF started-procedures table (Part 3 of 5)*

```
*        ENTRIES FOR SUBSYSTEM NAME "DB2P"
         DC    CL8'DB2PMSTR'       SYSTEM SERVICES PROCEDURE
         DC    CL8'SYSDSPD'        USERID
         DC    CL8'DB2PROD'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DB2PDBM1'     DATABASE SERVICES PROCEDURE
         DC    CL8'SYSDSPD'        USERID
         DC    CL8'DB2PROD'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DB2PDIST'       DDF PROCEDURE
         DC    CL8'SYSDSPD'        USERID
         DC    CL8'DB2PROD'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'DB2PSPAS'       STORED PROCEDURES PROCEDURE
         DC    CL8'SYSDSPD'        USERID
         DC    CL8'DB2PROD'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
```

*Figure 22. Sample job to reassemble the RACF started-procedures table (Part 4 of 5)*

```
*        OPTIONAL ENTRIES FOR CICS AND IMS CONTROL REGION
         DC    CL8'CICSSYS'        CICS PROCEDURE NAME
         DC    CL8'CICS'           USERID
         DC    CL8'CICSGRP'        GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
         DC    CL8'IMSCNTL'   IMS CONTROL REGION PROCEDURE
         DC    CL8'IMS'            USERID
         DC    CL8'IMSGRP'         GROUP NAME
         DC    X'00'               NO PRIVILEGED ATTRIBUTE
         DC    XL7'00'             RESERVED BYTES
ENDTABLE DS    0D
         END
```

*Figure 22. Sample job to reassemble the RACF started-procedures table (Part 5 of 5)*

## Add RACF groups

The details of this step depend on the groups you have defined. To add the user DB2OWNER, issue the following RACF command:

```
ADDUSER  DB2OWNER  CLAUTH(DSNR USER) UACC(NONE)
```

That gives class authorization to DB2OWNER for DSNR and USER. DB2OWNER can add users to RACF and issue the RDEFINE command to define resources in class DSNR. DB2OWNER has control over and responsibility for the entire DB2 security plan in RACF.

The RACF group SYS1 already exists. To add group DB2 and make DB2OWNER its owner, issue the following RACF command:

```
ADDGROUP DB2 SUPGROUP(SYS1) OWNER(DB2OWNER)
```

To connect DB2OWNER to group DB2 with the authority to create new subgroups, add users, and manipulate profiles, issue the following RACF command:

```
CONNECT DB2OWNER GROUP(DB2) AUTHORITY(JOIN) UACC(NONE)
```

To make DB2 the default group for commands issued by DB2OWNER, issue the following RACF command:

```
ALTUSER DB2OWNER DFLTGRP(DB2)
```

To create the group DB2USER and add five users to it, issue the following RACF commands:

```
ADDGROUP DB2USER SUPGROUP(DB2)
ADDUSER (USER1 USER2 USER3 USER4 USER5) DFLTGRP(DB2USER)
```

To define a user to RACF, use the RACF ADDUSER command. That invalidates the current password. You can then log on as a TSO user to change the password.

***DB2 considerations when using RACF groups:***

- When a user is newly connected to, or disconnected from, a RACF group, the change is not effective until the next logon. Therefore, before using a new group name as a secondary authorization ID, a TSO user must log off and log on, or a CICS or IMS user must sign on again.
- A user with the SPECIAL, JOIN, or GROUP-SPECIAL RACF attribute can define new groups with any name that RACF accepts and can connect any user to them. Because the group name can become a secondary authorization ID, you should control the use of those RACF attributes.
- Existing RACF group names can duplicate existing DB2 authorization IDs. That is unlikely, because a group name cannot be the same as a user name, and authorization IDs that are known to DB2 are usually user IDs that are known to RACF. However, if you create a table with an owner name that happens to be a RACF group name, and you use the IBM-supplied sample connection exit routine, any TSO user with the group name as a secondary ID has ownership privileges on the table. You can prevent that situation by designing the connection exit routine to stop unwanted group names from being passed to DB2. For example, in CICS, if the RCT specifies AUTH=TXID, ensure that the transaction identifier is not a RACF group; if it is, any user that is connected to the group has the same privileges as the transaction code.

## Permit access for users and groups

In this scenario, DB2OWNER is authorized for class DSNR, owns the profiles, and has the right to change them. The next commands let users that are members of the group DB2USER, and the system administrators and operators, to be TSO users, run batch jobs, and run DB2 utilities on the three systems: DSN, DB2P, and DB2T. The ACCESS(READ) operand allows use of DB2 without the ability to manipulate profiles.

```
PERMIT  DSN.BATCH  CLASS(DSNR) ID(DB2USER)  ACCESS(READ)
PERMIT  DB2P.BATCH CLASS(DSNR) ID(DB2USER)  ACCESS(READ)
PERMIT  DB2T.BATCH CLASS(DSNR) ID(DB2USER)  ACCESS(READ)
```

***IMS and CICS:*** You want the IDs for attaching systems to use the appropriate access profile. For example, to let the IMS user ID use the access profile for IMS on system DB2P, issue the following RACF command:

```
PERMIT  DB2P.MASS  CLASS(DSNR) ID(IMS)  ACCESS(READ)
```

To let the CICS group ID use the access profile for CICS on system DB2T, issue the following RACF command:

```
PERMIT  DB2T.SASS   CLASS(DSNR) ID(CICSGRP)  ACCESS(READ)
```

***Default IDs for installation authorities:*** When DB2 is installed, IDs are named to have special authorities—one or two IDs for SYSADM and one or two IDs for

SYSOPR. Those IDs can be connected to the group DB2USER; if they are not, you need to give them access. The next command permits the default IDs for the SYSADM and SYSOPR authorities to use subsystem DSN through TSO:

```
PERMIT  DSN.BATCH  CLASS(DSNR) ID(SYSADM,SYSOPR) ACCESS(READ)
```

IDs also can be group names.

***Secondary IDs:*** You can use secondary authorization IDs to define a RACF group. After you define the RACF group, you can assign privileges to it that are shared by multiple primary IDs. For example, suppose that DB2OWNER wants to create a group GROUP1 and give the ID USER1 administrative authority over it. USER1 should be able to connect other existing users to the group. To create the group, DB2OWNER issues this RACF command:

```
ADDGROUP  GROUP1  OWNER(USER1) DATA('GROUP FOR DEPT. G1')
```

To let the group connect to the DSN system through TSO, DB2OWNER issues this RACF command:

```
PERMIT DSN.BATCH CLASS(DSNR) ID(GROUP1) ACCESS(READ)
```

USER1 can now connect other existing IDs to the group GROUP1, using RACF CONNECT commands like this:

```
CONNECT (USER2 EPSILON1 EPSILON2) GROUP(GROUP1)
```

If you add or update secondary IDs for CICS transactions, you must start and stop the CICS attachment facility to ensure that all threads sign on and get the correct security information.

***Allowing users to create data sets:*** “Chapter 14. Auditing” on page 219 recommends using RACF to protect the data sets that store DB2 data. If you use that method, then when you create a new group of DB2 users, you might want to connect it to a group that can create data sets. Looking ahead to the methods of the next chapter, to allow USER1 to create and control data sets, DB2OWNER creates a generic profile and permits complete control to USER1, and also to DB2 (through SYSDSP) and to the four administrators.

```
ADDSD  'DSNC710.DSNDBC.ST*' UACC(NONE)

PERMIT 'DSNC710.DSNDBC.ST*'
       ID(USER1 SYSDSP SYSAD1 SYSAD2 SYSOP1 SYSOP2) ACCESS(ALTER)
```

***Allowing access from remote requesters:*** The recommended way of controlling access from remote requesters is to use the DSNR RACF class with a PERMIT command to access the distributed data address space (such as DSN.DIST). For example, the following RACF commands let the users in the group DB2USER to access DDF on the DSN subsystem. These DDF requests can originate from any partner in the network.

```
PERMIT  DSN.DIST   CLASS(DSNR) ID(DB2USER)  ACCESS(READ)
```

If you want to ensure that a specific user be allowed access only when the request originates from a specific LU name, you can use WHEN(APPCPORT) on the PERMIT command. For example, to permit access to DB2 distributed processing on subsystem DSN when the request comes from USER5 at LUNAME equal to NEWYORK, issue the following RACF command:

```
PERMIT DSN.DIST CLASS(DSNR) ID(USER5) ACCESS(READ) +
       WHEN(APPCPORT(NEWYORK))
```

For connections coming in through TCP/IP, you must use TCPIP as the APPCPORT name, as shown here:

```
PERMIT DSN.DIST CLASS(DSNR) ID(USER5) ACCESS(READ) +
      WHEN(APPCPORT(TCPIP))
```

If the RACF APPCPORT class is active on your system, and a resource profile for the requesting LU name already exists, you must permit READ access to the APPCPORT resource profile for the user IDs that DB2 uses, even when you are using the DSNR resource class. Similarly, if you are using the RACF APPL class and that class is restricting access to the local DB2 LU name or generic LU name, you must permit READ access to the APPL resource for the user IDs that DB2 uses.

## Establishing RACF protection for stored procedures

This section summarizes the procedures that you can follow for establishing RACF protection for stored procedures that run on your DB2 subsystem.

This section contains the following procedures:

- “Step 1: Control access by using the attachment facilities (required)”
- “Step 2: Control access to WLM (optional)” on page 210
- “Step 3: Control access to non-DB2 resources (optional)” on page 211.

### Step 1: Control access by using the attachment facilities (required)

The user ID that is associated with the DB2-established address space must be authorized to run the DB2 call attachment facility. It must be associated with the *ssnm*.BATCH profile, as described in “Define the names of protected access profiles” on page 200.

The user ID that is associated with the WLM-established stored procedures address space must be authorized to run Recoverable Resource Manager Services attachment facility (RRSAF) and is associated with the *ssnm*.RRSAF profile.

Control access to the DB2 subsystem through RRSAF by performing the following steps:

1. If you have not already established a profile for controlling access from the RRS attachment as described in “Define the names of protected access profiles” on page 200, define *ssnm*.RRSAF in the DSNR resource class with a universal access authority of NONE, as shown in the following command:

   ```
   RDEFINE DSNR (DB2P.RRSAF  DB2T.RRSAF) UACC(NONE)
   ```

2. Activate the resource class; use the following command:

   ```
   SETROPTS RACLIST(DSNR) REFRESH
   ```

3. Add user IDs that are associated with the stored procedures address spaces to the RACF Started Procedures Table, as shown in this example:

   ```
   ⋮
            DC    CL8'DSNWLM'        WLM-ESTABLISHED S.P. ADDRESS SPACE
            DC    CL8'SYSDSP'        USERID
            DC    CL8'DB2SYS'        GROUP NAME
            DC    X'00'              NO PRIVILEGED ATTRIBUTE
            DC    XL7'00'            RESERVED BYTES
   ⋮
   ```

4. Allow read access to *ssnm*.RRSAF to the user ID that is associated with the stored procedures address space:

```
PERMIT  DB2P.RRSAF  CLASS(DSNR) ID(SYSDSP)  ACCESS(READ)
```

## Step 2: Control access to WLM (optional)

Optionally, you can control which address spaces can be WLM-established server address spaces that run stored procedures. To do this, use the *server* resource class, which WLM uses to identify valid address spaces to which work can be sent. If the server class is not defined or activated, any address space is allowed to connect to WLM as a server address space and to identify itself as a server address space that runs stored procedures.

To use the server resource class, perform the following steps:

1. Run a version of RACF in which the resource class SERVER is included as part of the predefined resource classes (RACF Version 2 Release 2 and subsequent releases).
2. Define a RACF profile for resource class SERVER, as follows:

   ```
   RDEFINE SERVER (DB2.ssnm.applenv)
   ```

   where *applenv* is the name of the application environment that is associated with the stored procedure. See "Assigning procedures and functions to WLM application environments" on page 875 for more information about application environments.

   Assume you want to define the following profile names:

   - DB2.DB2T.TESTPROC
   - DB2.DB2P.PAYROLL
   - DB2.DB2P.QUERY

   Use the following RACF command:

   ```
   RDEFINE SERVER (DB2.DB2T.TESTPROC DB2.DB2P.PAYROLL DB2.DB2P.QUERY)
   ```
3. Activate the SERVER resource class:

   ```
   SETROPTS RACLIST(SERVER) REFRESH
   ```
4. Permit read access to the server resource name to the user IDs that are associated with the stored procedures address space.

   ```
   PERMIT  DB2.DB2T.TESTPROC CLASS(SERVER) ID(SYSDSP) ACCESS(READ)
   PERMIT  DB2.DB2P.PAYROLL  CLASS(SERVER) ID(SYSDSP) ACCESS(READ)
   PERMIT  DB2.DB2P.QUERY    CLASS(SERVER) ID(SYSDSP) ACCESS(READ)
   ```

***Control of stored procedures in a WLM environment:*** Programs can be grouped together and isolated in different WLM environments based on application security requirements. For example, payroll applications might be isolated in one WLM environment, because they contain sensitive data, such as employee salaries.

To prevent users from creating a stored procedure in a sensitive WLM environment, DB2 invokes RACF to determine if the user is allowed to create stored procedures in the specified WLM environment. The WLM ENVIRONMENT keyword on the CREATE PROCEDURE statement identifies the WLM environment to use for running a given stored procedure. Attempts to create a procedure fail if the user is not properly authorized.

DB2 performs a resource authorization check using the DSNR RACF class as follows:

- In a DB2 data sharing environment, DB2 uses the following RACF resource name:

  ```
  db2_groupname.WLMENV.wlm_environment
  ```

- In a non-data sharing environment, DB2 checks the following RACF resource name:

  ```
  db2_subsytem_id.WLMENV.wlm_environment
  ```

You can use the RACF RDEFINE command to create RACF profiles that prevent users from creating stored procedures and user-defined functions in sensitive WLM environments. For example, you can prevent all users on DB2 subsystem DB2A (non-data sharing) from creating a stored procedure or user-defined function in the WLM environment named PAYROLL; to do this, use the following command:

```
RDEFINE DSNR (DB2A.WLMENV.PAYROLL) UACC(NONE)
```

The RACF PERMIT command authorizes a user to create a stored procedure or user-defined function in a WLM environment. For example, you can authorize a DB2 user (DB2USER1) to create stored procedures on DB2 subsystem DB2A (non-data sharing) in the WLM environment named PAYROLL:

```
PERMIT  DB2A.WLMENV.PAYROLL  CLASS(DSNR) ID(DB2USER1)  ACCESS(READ)
```

***Control of stored procedures in a DB2-established stored procedures address space:*** DB2 invokes RACF to determine if a user is allowed to create a stored procedures in a DB2-established stored procedures address space. The NO WLM ENVIRONMENT keyword on the CREATE PROCEDURE statement indicates that a given stored procedure will run in a DB2-established stored procedures address space. Attempts to create a procedure fail if the user is not authorized, or if there is no DB2-established stored procedures address space.

The RACF PERMIT command authorizes a user to create a stored procedure in a DB2-established stored procedures address space. For example, you can authorize a DB2 user (DB2USER1) to create stored procedures on DB2 subsystem DB2A in the stored procedures address space named DB2ASPAS:

```
PERMIT  DB2A.WLMENV.DB2ASPAS  CLASS(DSNR) ID(DB2USER1)  ACCESS(READ)
```

### Step 3: Control access to non-DB2 resources (optional)

With stored procedures that run in a DB2-established address space, the user ID of the stored procedures address space (from the started procedures table of RACF) is used to access non-DB2 resources such as IMS or CICS transactions, MVS/APPC conversations, or VSAM files.

With WLM-established address spaces, you can specify that access to non-DB2 resources is controlled by the authorization ID of the caller rather than that of the stored procedures address space. To do this, specify U in the EXTERNAL_SECURITY column of table SYSIBM.SYSROUTINES for the stored procedure.

When you specify U for EXTERNAL_SECURITY, a separate RACF environment is established for that stored procedure. Use EXTERNAL_SECURITY=U only when the caller must access resources outside of DB2. Figure 23 shows the user ID that is associated with each part of a stored procedure.

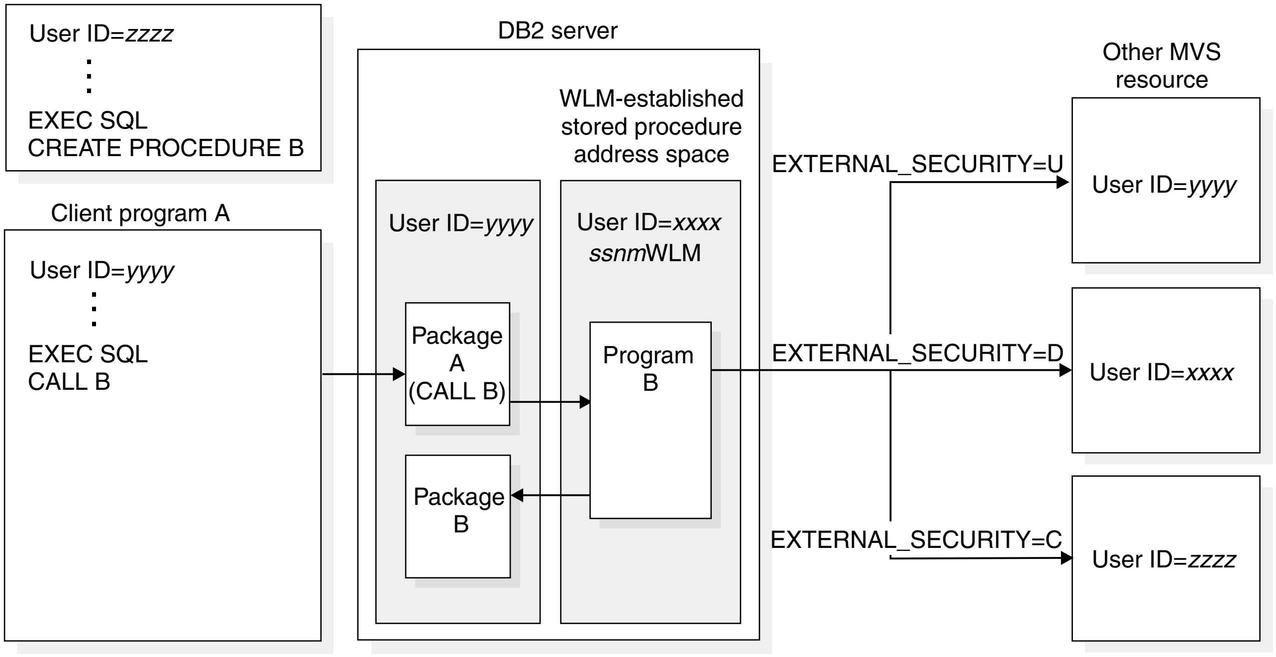


*Figure 23. Accessing non-DB2 resources from a stored procedure*

For WLM-established stored procedures address spaces, enable the RACF check for the caller's ID when accessing non-DB2 resources by performing the following steps:

1. Update the row for the stored procedure in table SYSIBM.SYSROUTINES with EXTERNAL_SECURITY=U.
2. Ensure that the ID of the stored procedure's caller has RACF authority to the resources.
3. For the best performance, cache the RACF profiles in the virtual look-aside facility (VLF) of MVS. Do this by specifying the following keywords in the COFVLF*xx* member of library SYS1.PARMLIB.

   ```
   CLASS NAME(IRRACEE)
   EMAJ(ACEE)
   ```

## Establishing RACF protection for TCP/IP

The ID that is associated with DB2's distributed address space must be authorized to use OS/390 UNIX services if your DB2 subsystem is going to send or accept any requests over TCP/IP. To do this, you must create an OMVS segment in the RACF user profile, as follows:

```
ALTUSER (SYSDSP) OMVS(UID(0))
```

To give root authority to the DDF address space, you must specify a UID of 0.

## Establishing Kerberos authentication through RACF

Kerberos security is a network security technology developed at the Massachusetts Institute of Technology. DB2 for OS/390 and z/OS can use Kerberos security services to authenticate remote users. With Kerberos security services, remote end users access DB2 for OS/390 and z/OS when they issue their Kerberos name and password. This same name and password is used for access throughout the network, so a separate MVS password to access DB2 for OS/390 and z/OS is not necessary.

The Kerberos security technology does not require passwords to flow in readable text, making it secure even for client/server environments. This flexibility is possible because Kerberos uses an authentication technology that uses encrypted tickets that contain authentication information for the end user.

DB2 for OS/390 and z/OS support for Kerberos security requires the OS/390 SecureWay Security Server (formerly known as RACF) and the Security Server Network Authentication and Privacy Service, or the functional equivalent. The Network Authentication and Privacy Service provides Kerberos support and relies on a security product, such as RACF, to provide registry support. The OS/390 Security Server allows administrators who are already familiar with RACF commands and RACF ISPF panels to define Kerberos configuration and principal information. For more information about using Kerberos security, see the *OS/390 SecureWay Security Server Network Authentication and Privacy Service Administration Guide*, *OS/390 Security Server (RACF) Security Administrator's Guide* and *OS/390 Security Server (RACF) Command Language Reference*.

Each remote user who is authenticated to DB2 by means of Kerberos authentication must be registered in RACF profiles.

1. Define the Kerberos realm to RACF. The name of the local realm must be supplied in the definition. You must also supply a Kerberos password for RACF to grant Kerberos ticket-granting tickets. Define a Kerberos realm with the following command:

   ```
   RDEFINE REALM KERBDFLT KERB(KERBNAME(localrealm) PASSWORD(mykerpw)
   ```

2. Define local principals to RACF. The RACF passwords must be changed before the principals become active Kerberos users. Define a Kerberos principal with the following commands:

   ```
   AU RONTOMS KERB(KERBNAME(rontoms))
   ALU RONTOMS PASSWORD(new1pw) NOEXPIRE
   ```

3. Map foreign Kerberos principals by defining KERBLINK profiles to RACF with a command similar to the following command:

   ```
   RDEFINE KERBLINK /.../KERB390.ENDICOTT.IBM.COM/RWH APPLDATA('RONTOMS')
   ```

   You must also define a principal name for the user ID used in the ssnmDIST started task address space. This step is required because the ssnmDIST address space must have the RACF authority to use its SAF ticket parsing service.

   ```
   ALU SYSDSP PASSWORD(pw) NOEXPIRE KERB(KERBNAME(SYSDSP))
   ```

   In this example, the user ID that is used for the ssnmDIST started task address space is SYSDSP. See “Define RACF user IDs for DB2 started tasks” on page 203 for more information, including how to determine the user ID for the ssnmDIST started task.

4. Define foreign Kerberos authentication servers to the local Kerberos authentication server using REALM profiles. You must supply a password for the key to be generated. REALM profiles define the trust relationship between the local realm and the foreign Kerberos authentication servers. PASSWORD is a required keyword, so all REALM profiles have a KERB segment. The command is similar to the following command:

   ```
   RDEFINE REALM /.../KERB390.ENDICOTT.IBM.COM/KRBTGT/KER2000.ENDICOTT.IBM.COM +
   KERB(PASSWORD(realm0pw))
   ```

The OS/390 SecureWay Kerberos Security Server rejects ticket requests from users with revoked or expired passwords, so plan password resets that use a method avoiding a password change at a subsequent logon. For example, use the TSO logon panel the PASSWORD command without the ID operand specified, or the ALTUSER command with NOEXPIRE specified.

**Data sharing environment:** Data sharing Sysplex environments that use Kerberos security must have a Kerberos Security Server instance running on each system in the Sysplex. The instances must either be in the same realm and share the same RACF database, or have different RACF databases and be in different realms.

## Other methods of controlling access

You can also help control access to DB2 from within IMS or CICS.

- IMS security

  IMS terminal security lets you limit the entry of a transaction code to a particular logical terminal (LTERM) or group of LTERMs in the system. To protect a particular program, you can authorize a transaction code to be entered only from any terminal on a list of LTERMs. Alternatively, you can associate each LTERM with a list of the transaction codes that a user can enter from that LTERM. IMS then passes the validated LTERM name to DB2 as the initial primary authorization ID.

- CICS security

  CICS transaction code security works with RACF to control the transactions and programs that can access DB2. Within DB2, you can use the ENABLE and DISABLE options of the bind operation to limit access to specific CICS subsystems.

# Chapter 13. Protecting data sets

To fully protect the data in DB2, you must take steps to ensure that no other process has access to the data sets in which DB2 data resides.

**Recommendation:** Use RACF, or a similar external security system, to control access to the data sets just as it controls access to the DB2 subsystem. "Controlling data sets through RACF" explains how to create RACF profiles for data sets and allow their use through DB2.

## Controlling data sets through RACF

Assume that the RACF groups DB2 and DB2USER, and the RACF user ID DB2OWNER, have been set up for DB2 IDs, as described under "Defining DB2 resources to RACF" on page 200 (and shown in Figure 20 on page 199). Given that setting, the examples that follow show you how to:

- Add RACF groups to control data sets that use the default DB2 qualifiers
- Create generic profiles for different types of DB2 data sets and permit their use by DB2 started tasks
- Permit use of the profiles by specific IDs
- Allow certain IDs to create data sets.

## Adding groups to control DB2 data sets

The default high-level qualifier for data sets containing DB2 databases and recovery logs is DSNC710; for distribution, target, SMP, and other installation data sets, it is DSN710. DB2OWNER can create groups that control those data sets, by issuing the following commands:

```
ADDGROUP DSNC710 SUPGROUP(DB2) OWNER(DB2OWNER)
ADDGROUP DSN710 SUPGROUP(DB2) OWNER(DB2OWNER)
```

## Creating generic profiles for data sets

DB2 uses specific names to identify data sets for special purposes. In "Define RACF user IDs for DB2 started tasks" on page 203, SYSDSP is the RACF user ID for DB2 started tasks. DB2OWNER can issue the following commands to create generic profiles for the data set names and give complete control over the data sets to DB2 started tasks:

- For active logs, issue the following commands:

  ```
  ADDSD  'DSNC710.LOGCOPY*' UACC(NONE)
  PERMIT 'DSNC710.LOGCOPY*' ID(SYSDSP) ACCESS(ALTER)
  ```

- For archive logs, issue the following commands:

  ```
  ADDSD  'DSNC710.ARCHLOG*' UACC(NONE)
  PERMIT 'DSNC710.ARCHLOG*' ID(SYSDSP) ACCESS(ALTER)
  ```

- For bootstrap data sets, issue the following commands:

  ```
  ADDSD  'DSNC710.BSDS*'    UACC(NONE)
  PERMIT 'DSNC710.BSDS*'    ID(SYSDSP) ACCESS(ALTER)
  ```

- For table spaces and index spaces, issue the following commands:

  ```
  ADDSD  'DSNC710.DSNDBC.*' UACC(NONE)
  PERMIT 'DSNC710.DSNDBC.*' ID(SYSDSP) ACCESS(ALTER)
  ```

- For installation libraries, issue the following commands:

  ```
  ADDSD 'DSN710.*'          UACC(READ)
  ```

Started tasks do not need control.

- For other general data sets, issue the following commands:

  ```
  ADDSD  'DSNC710.*'       UACC(NONE)
  PERMIT 'DSNC710.*'       ID(SYSDSP) ACCESS(ALTER)
  ```

Although all of those commands are not absolutely necessary, the sample shows how you can create generic profiles for different types of data sets. Some parameters, such as universal access, could vary among the types. In the example, installation data sets (DSN710.*) are universally available for read access.

If you use generic profiles, specify NO on installation panel DSNTIPP for ARCHIVE LOG RACF, or you might get an MVS error when DB2 tries to create the archive log data set. If you specify YES, DB2 asks RACF to create a separate profile for each archive log that is created, which means you cannot use generic profiles for these data sets.

To protect VSAM data sets, use the cluster name. You do not need to protect the data component names, because the cluster name is used for RACF checking.

***Access by stand-alone DB2 utilities:*** The following DB2 utilities access objects outside of DB2 control:

- DSN1COPY and DSN1PRNT: table space and index space data sets
- DSN1LOGP: active logs, archive logs, and bootstrap data sets
- DSN1CHKR: DB2 directory and catalog table spaces
- Change Log Inventory (DSNJU003) and Print Log Map (DSNJU004): bootstrap data sets

The Change Log Inventory and Print Log Map are batch jobs that are protected by the USER and PASSWORD options on the JOB statement. To provide a value for the USER option, for example SVCAID, issue the following commands:

- For DSN1COPY:

  ```
  PERMIT 'DSNC710.*' ID(SVCAID) ACCESS(CONTROL)
  ```

- For DSN1PRNT:

  ```
  PERMIT 'DSNC710.*' ID(SVCAID) ACCESS(READ)
  ```

- For DSN1LOGP:

  ```
  PERMIT 'DSNC710.LOGCOPY*' ID(SVCAID) ACCESS(READ)
  PERMIT 'DSNC710.ARCHLOG*' ID(SVCAID) ACCESS(READ)
  PERMIT 'DSNC710.BSDS*'    ID(SVCAID) ACCESS(READ)
  ```

- For DSN1CHKR:

  ```
  PERMIT 'DSNC710.DSNDBDC.*' ID(SVCAID) ACCESS(READ)
  ```

- For change log inventory:

  ```
  PERMIT 'DSNC710.BSDS*' ID(SVCAID) ACCESS(CONTROL)
  ```

- For print log map:

  ```
  PERMIT 'DSNC710.BSDS*' ID(SVCAID) ACCESS(READ)
  ```

The level of access depends on the intended use, not on the type of data set (VSAM KSDS, VSAM linear, or sequential). For update operations, ACCESS(CONTROL) is required; for read-only operations, ACCESS(READ) is sufficient.

You can use RACF to permit programs, rather than user IDs, to access objects. When you use RACF in this manner, IDs that are not authorized to access the log data sets might be able to do so by running the DSN1LOGP utility. Permit access to database data sets through DSN1PRNT or DSN1COPY.

## Permitting DB2 authorization IDs to use the profiles

Authorization IDs with installation SYSADM or installation SYSOPR authority need access to most DB2 data sets. (For a list of the privileges that go with those authorities, see “Explicit privileges and authorities” on page 104.) The following command adds the two default IDs that have the SYSADM and SYSOPR authorities if no other IDs are named when DB2 is installed:

```
ADDUSER (SYSADM SYSOPR)
```

The next two commands connect those IDs to the groups that control data sets, with the authority to create new RACF database profiles. The ID that has Installation SYSOPR authority (SYSOPR) does not need that authority for the installation data sets.

```
CONNECT (SYSADM SYSOPR)  GROUP(DSNC710) AUTHORITY(CREATE) UACC(NONE)
CONNECT (SYSADM)         GROUP(DSN710)  AUTHORITY(CREATE) UACC(NONE)
```

The next set of commands gives the IDs complete control over DSNC710 data sets. The system administrator IDs also have complete control over the installation libraries. Additionally, you can give the system programmer IDs the same control.

```
PERMIT 'DSNC710.LOGCOPY*' ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC710.ARCHLOG*' ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC710.BSDS*'    ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC710.DSNDBC.*' ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC710.*'        ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSN710.*'         ID(SYSADM)        ACCESS(ALTER)
```

## Allowing DB2 authorization IDs to create data sets

The next command connects several IDs, which are already connected to the DB2USER group, to group DSNC710 with CREATE authority:

```
CONNECT (USER1 USER2 USER3 USER4 USER5)
        GROUP(DSNC710) AUTHORITY(CREATE) UACC(NONE)
```

Those IDs can now explicitly create data sets whose names have DSNC710 as the high-level qualifier. Any such data sets that are created by DB2 or by these RACF user IDs are protected by RACF. Other RACF user IDs are prevented by RACF from creating such data sets.

If no option is supplied for PASSWORD on the ADDUSER command that adds those IDs, the first password for the new IDs is the name of the default group, DB2USER. The first time that the IDs sign on, they all use that password, but must change it during their first session.

# Chapter 14. Auditing

This chapter provides answers to some fundamental auditing questions. Foremost among them are these:

1. Who is privileged to access what objects?
2. Who has actually accessed the data?

Answers to the first question are found in the DB2 catalog, which is a primary audit trail for the DB2 subsystem. Most of the catalog tables describe the DB2 objects, such as tables, views, table spaces, packages, and plans. Several other tables (every one with the characters “AUTH” in its name) hold records of every grant of a privilege or authority on different types of object. Every record of a grant contains the name of the object, the ID that received the privilege, the ID that granted it, the time of the grant, and other information.

You can retrieve data from catalog tables by writing SQL queries. For examples, see “Finding catalog information about privileges” on page 152.

Answers to the second question are revealed by the audit trace, another primary audit trail for DB2. The trace can record changes in authorization IDs for a security audit and changes that are made to the structure of data (such as dropping a table) or data values (such as updating or inserting records) for an audit of data access. The trace can also audit access attempts by unauthorized IDs, the results of GRANT and REVOKE statements, the mapping of Kerberos security tickets to RACF IDs, and other activities of interest to auditors.

This chapter also answers these auditing questions:

“Other sources of audit information” on page 225
“What security measures are in force?” on page 225
“What helps ensure data accuracy and consistency?” on page 226
“How can I tell that data is consistent?” on page 229
“How can DB2 recover data after failures?” on page 231
“How can I protect the software?” on page 232
“How can I ensure efficient usage of resources?” on page 232.

## How can I tell who has accessed the data?

The information in this section, up to “Other sources of audit information” on page 225 is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

The DB2 audit trace can tell who has accessed data. When started, the audit trace creates records of actions of certain types and sends them to a named destination. As with other DB2 traces, you can choose, by options of the audit trace:

- Categories of events to trace
- Particular authorization IDs or plan IDs to audit
- Ways to start and stop the trace
- Destinations for audit records

You can also choose whether to audit a table, by specifying an option of the CREATE and ALTER statements.

## Options of the audit trace

You specify most TRACE options when you issue the commands START TRACE and STOP TRACE.

### The role of authorization IDs

In general, audit trace records identify a process by its primary authorization ID. The value is recorded both before and after invocation of an authorization exit routine; therefore, you can identify a change. The exception to this is if a primary ID has been translated many times. For example, the translated ID at the requesting site might be unknown to the server. In that case, you cannot use the primary ID to gather all audit records for a user that accesses remote data. The AUTHCHG record also shows the values of all secondary authorization IDs that are established by an exit routine. See "Audit class descriptions", Audit Class 7, for a description of the AUTHCHG record.

With the trace, you can also determine which primary ID is responsible for the action of a secondary ID, when that information might not appear in the catalog. For example, suppose that the user with primary ID SMITHJ sets the current SQL ID to TESTGRP, in order to grant privileges over the table TESTGRP.TABLE01 to another user. The DB2 catalog records the grantor of the privileges as TESTGRP; the audit trace, however, shows that the grant statement was issued by SMITHJ.

***Use of exit routines:*** Because the trace identifies a process by its primary ID, consider carefully the consequences of altering that ID by an exit routine. If the primary ID identifies a unique user, individual accountability is possible. However, if several users share the same primary ID, say a RACF group name, then you cannot tell which of them issued a particular GRANT statement or ran a particular application plan.

### Auditing classes of events

The audit trace does not record everything. The actual changed data is not recorded (it is recorded in the log). If an agent or transaction accesses a table more than once in a single unit of recovery, only the first access is recorded, and then only if the audit trace is started for the appropriate class of events.

Some utilities are not audited. The first access of a table by LOAD is audited, but access by COPY, RECOVER, and REPAIR is not. Access by stand-alone utilities, such as DSN1CHKR and DSN1PRNT, is not audited.

This auditing coverage is consistent with the goal of providing a moderate volume of data with a low impact on performance. However, when you choose classes of events to audit, consider that you might ask for more data than you care to process.

### Audit class descriptions

When you start the trace, you choose events to audit by giving one or more numbers, to identify classes of events. The trace records are limited to 5000 bytes, so descriptions that contain long SQL statements might be truncated. The available classes and the events they include are as follows:

| Audit | Class events traced |
|---|---|
| 1 | Access attempts that DB2 denies because of inadequate authorization. This class is the default. |
| 2 | Explicit GRANT and REVOKE statements and their results. The class does not include implicit grants and revokes. |
| 3 | CREATE, ALTER, and DROP operations affecting audited tables, and their |

results. The class includes the dropping of a table caused by DROP TABLESPACE or DROP DATABASE and the creation of a table with AUDIT CHANGES or AUDIT ALL. ALTER TABLE statements are audited only when they change the AUDIT option for the table.

**4** Changes to audited tables. Only the first attempt to change a table, within a unit of recovery, is recorded. (If the agent or the transaction issues more than one COMMIT statement, the number of audit records increases accordingly.) The changed data is not recorded, only the attempt to make a change. If the change is not successful and is rolled back, the audit record remains; it is not deleted. This class includes access by the LOAD utility.

Accesses to a dependent table that are caused by attempted deletions from a parent table are also audited. The audit record is written even if the delete rule is RESTRICT, which prevents the deletion from the parent table. The audit record is also written when the rule is CASCADE or SET NULL, which can result in deletions cascading to the dependent table.

**5** All read accesses to tables that are identified as AUDIT ALL. As in class 4, only the first access within a DB2 unit of recovery is recorded, and references to a parent table are audited.

**6** The bind of static and dynamic SQL statements of the following types:

- INSERT, UPDATE, DELETE, CREATE VIEW, and LOCK TABLE statements for audited tables. Except for the values of host variables, the entire SQL statement is contained in the audit record.
- SELECT statements to tables that are identified as AUDIT ALL. Except for the values of host variables, the entire SQL statement is contained in the audit record.

**7** Assignment or change of an authorization ID, through an exit routine (default or user-written) or a SET CURRENT SQLID statement, through either an outbound or inbound authorization ID translation, or because the ID is being mapped to a RACF ID from a Kerberos security ticket.

**8** The start of a utility job, and the end of each phase of the utility.

**9** Various types of records that are written to IFCID 0146 by the IFI WRITE function.

## Auditing specific IDs

As with other DB2 traces, you can start an audit trace for a particular plan name, a particular primary authorization ID, or a combination of the two. For examples, see “DB2 trace” on page 1033. Having audit traces on at all times can be useful for IDs with SYSADM authority, for example, because they have complete access to every table. If you have a network of DB2 subsystems, you might need to trace multiple authorization IDs for those users whose primary authorization ID are translated several times.

## Starting and stopping the audit trace

You can cause an audit trace to start automatically whenever DB2 is started by making a choice on the panel DSNTIPN when DB2 is installed. Set AUDIT TRACE to NO, YES, or a list of audit trace classes.

- Use * (an asterisk) to provide a complete audit trace.
- Use NO, the default, if you do not want an audit trace to start automatically.
- Use YES to start a trace automatically for the default class (class 1: access denials) and the default destination (the SMF data set).

- Use a list of audit trace classes (for example, 1,3,5) to start a trace automatically for those classes. It uses the default destination.

***The START TRACE command:*** As with other DB2 traces, you can start an audit trace at any time with the -START TRACE command. You can choose the audit classes to trace and the destination for trace records. You can also include an identifying comment. For example, this command starts an audit trace for classes 4 and 6 with distributed activity:

```
-START TRACE (AUDIT) CLASS (4,6) DEST (GTF) LOCATION (*)
  COMMENT ('Trace data changes; include text of dynamic DML statements.')
```

***The STOP TRACE command:*** You can have several different traces running at the same time, including more than one audit trace. One way to stop a particular trace is to issue the -STOP TRACE command with the same options that were used for -START TRACE (or enough of them to identify a particular trace). For example, this command stops the trace that the last example started:

```
-STOP TRACE (AUDIT) CLASS (4,6) DEST (GTF)
```

If you have not saved the text of the command, it might be simpler to find out the identifying trace number and stop the trace by number. Use -DISPLAY TRACE to find the number. For example, -DISPLAY TRACE (AUDIT) might return a message something like this:

```
TNO  TYPE    CLASS         DEST     QUAL
01   AUDIT   01            SMF      NO
02   AUDIT   04,06         GTF      YES
```

The message indicates that two audit traces are active. Trace 1 traces events in class 1 and sends records to the SMF data set; it can be a trace that starts automatically whenever DB2 is started. Trace 2 traces events in classes 4 and 6 and sends records to GTF; the trace that the last example started can be identified like that.

You can stop either trace by its identifying number (TNO). Use commands like these:

```
-STOP TRACE AUDIT TNO(1)
-STOP TRACE AUDIT TNO(2)
```

### Considerations for distributed data

The DB2 audit trace audits any access to your data, whether the request is from a remote location or your local DB2. The authorization ID on a trace record for a remote request is the ID that is the final result of any outbound translation, inbound translation, or activity of an authorization exit routine; that is, it is the same ID to which you have granted access privileges for your data.

Requests from your location to a remote DB2 are audited only if an audit trace is active at the remote location. The output from the trace appears only in the records at that location.

## Auditing a specific table

The auditing described in this chapter takes place only when the audit trace is on and, where it relates to tables, only for tables you specifically choose to audit. Access to auxiliary tables cannot be audited. You do not create catalog tables and cannot alter them; therefore, you cannot audit the catalog tables.

To choose to audit a table, use the AUDIT clause in the CREATE TABLE or ALTER TABLE statement. For example, the department table is audited whenever the audit trace is on, if you create it with this statement:

```
CREATE TABLE DSN8710.DEPT
      (DEPTNO    CHAR(3)          NOT NULL,
       DEPTNAME  VARCHAR(36)      NOT NULL,
       MGRNO     CHAR(6)                  ,
       ADMRDEPT  CHAR(3)          NOT NULL,
       LOCATION  (CHAR16)                 ,
       PRIMARY KEY (DEPTNO)               )
  IN DSN8D71A.DSN8S71D
  AUDIT CHANGES;
```

That example changes the one under “Department table (DSN8710.DEPT)” on page 884 only by adding the last line. The option CHANGES causes the table to be audited for accesses that would insert, update, or delete data (trace class 4).

To cause the table to be audited for read accesses also (class 5), issue the following statement:

```
ALTER TABLE DSN8710.DEPT
  AUDIT ALL;
```

The statement is effective regardless of whether the table had been chosen for auditing before.

To prevent all auditing of the table, issue the following statement:

```
ALTER TABLE DSN8710.DEPT
  AUDIT NONE;
```

For CREATE TABLE, the default audit option is NONE. For ALTER TABLE, no default exists; if you do not use the AUDIT clause in an ALTER TABLE statement, the audit option for the table is unchanged.

When CREATE TABLE or ALTER TABLE statements affect the auditing of a table, those statements can themselves be audited; but the results of those operations are in audit class 3, not in class 4 or 5. Use audit class 3 to determine whether auditing was turned off for some table for an interval of time.

If an ALTER TABLE statement turns auditing on or off for a specific table, plans and packages that use the table are invalidated and must be rebound. Changing the auditing status does not affect plans, packages, or dynamic SQL statements that are currently running. The change is effective only for plans, packages, or dynamic SQL statements that begin running after the ALTER TABLE statement has completed.

## Using audit records

Considerations for preparing the System Management Facility (SMF) or Generalized Trace Facility (GTF) for accepting audit trace records are the same as for performance or accounting trace records. See “Recording SMF trace data” on page 1037 and “Recording GTF trace data” on page 1039 for information. The records are of SMF type 102, as are performance trace records.

All DB2 trace records are identified by IFCIDs. For instructions on interpreting trace output and mapping records for the IFCIDs, see “Appendix D. Interpreting DB2 trace output” on page 981. The IFCIDs for each trace class are listed with the description of the START TRACE command in Chapter 2 of *DB2 Command Reference*.

If you send trace records to SMF (the default), data might be lost in the following circumstances:

- SMF fails while DB2 continues running.
- An unexpected abend (such as a TSO interrupt) occurs while DB2 is transferring records to SMF.

In those circumstances, SMF records the number of records that are lost. MVS provides an option to stop the system rather than to lose SMF data.

## Reporting the records

Among other things, the audit trace records can indicate:

- The ID that initiated the activity
- The LOCATION of the ID that initiated the activity (if the access was initiated from a remote location)
- The type of activity and the time the activity occurred
- The DB2 objects that were affected
- Whether access was denied
- Who owns a particular plan and package

To extract, format, and print the records, you can use any of the following methods:

- Use DB2 PM. See "DB2 Performance Monitor (DB2 PM)" on page 1039 for more information.
- Write your own application program to access the SMF data.
- Use the instrumentation facility interface (IFI) as an online resource to pull audit records. For more information on using the IFI, see "Appendix E. Programming for the Instrumentation Facility Interface (IFI)" on page 997.

## Suggestions for reports

If you regularly start the audit trace for all classes, you accumulate data from which to draw reports like these:

- Usage of sensitive data

  You should probably define tables that contain sensitive data, such as employee salary records, with the AUDIT ALL option. You can report usage by table and by authorization ID,[7] to look for access by unusual IDs, at unusual times, or of unexpected types. You should also record any ALTER or DROP operations that affect the data. Use audit classes 3, 4, and 5.
- Grants of critical privileges

  Carefully monitor IDs with special authorities, such as SYSADM and DBADM, and with explicit privileges over sensitive data, such as an update privilege on records of accounts payable. A query of the DB2 catalog can show who holds such a privilege at a particular time. The audit records can reveal whether the privilege was granted and then revoked in a period of time. Use audit class 2.
- Unsuccessful access attempts

  Investigate all unsuccessful access attempts; some of those are only user errors, but others can be attempts to violate security. If you have sensitive data, always use trace audit class 1. You can report by table or by authorization ID.[7]

[7] For embedded SQL, the audited ID is the primary authorization ID of the person who bound the plan or package. For dynamic SQL, the audited ID is the primary authorization ID.

## Other sources of audit information

As well as the audit trace, other DB2 traces are also available. You can read about the accounting, statistics, and performance traces in “DB2 trace” on page 1033. DB2PM is useful to print reports of those traces, too; see “DB2 Performance Monitor (DB2 PM)” on page 1039.

Although the recovery log is not an all-purpose log, it can be useful for auditing. Information from the log can be printed using the DSN1LOGP utility. For example, the summary report can show which table spaces were updated within the range of the log that was scanned. The REPORT utility can indicate what log information is available and where it is located. For information on running DSN1LOGP and REPORT, see *DB2 Utility Guide and Reference*.

Image copies of table spaces are generated during typical recovery procedures. You can inspect those copies, or use them with the RECOVER utility to recover a table space to a particular point in time, which can help you narrow the time period in which a particular change was made. For guidance in using COPY and RECOVER, see “Chapter 21. Backing up and recovering databases” on page 373.

The MVS console log contains messages about exceptional conditions encountered during DB2 operation. Inspect it for symptoms of problems.

## What security measures are in force?

As an auditor, you are interested in the privileges and authorities that are associated with IDs in the DB2 subsystem. Read “Chapter 10. Controlling access to DB2 objects” on page 103.

A first step might be to see that DB2 authorization checking is actually in operation—it can be disabled. Follow the instructions for changing DB2 installation parameters that are given in “The Update Process” in Part 2 of *DB2 Installation Guide*. Without changing anything, look at panel DSNTIPP. If the value of USE PROTECTION is YES, DB2 checks privileges and authorities before permitting any activity.

To see what IDs hold particular privileges, look at the DB2 catalog. You can write appropriate SQL queries. Instructions are given in “Finding catalog information about privileges” on page 152.

The audit trace, described above, should be running to check access attempts on sensitive data. To see that the trace is running, display the status of the trace by the command DISPLAY TRACE(AUDIT).

Some authorization IDs you encounter are probably group IDs, to which many individual IDs can be connected. To see what IDs are connected to a group, you need a report from RACF, or from whatever external security system you are using. Similar reports can tell you what IDs have the required privileges to use DB2 data sets and other resources. For instructions on obtaining such reports, you need the documentation from the external security system; such as, *OS/390 Security Server (RACF) System Programmer's Guide*.

Data definition control is another security measure that provides additional constraints to existing authorization checks. With it, you control how specific plans or collections of packages can use SQL data definition (DDL) statements. Read “Chapter 11. Controlling access through a closed application” on page 157 for a description of this function. To determine if the control is active, look at option 1 on panel DSNTIPZ. To determine how DDL statements are controlled, see installation panel DSNTIPZ in Part 2 of *DB2 Installation Guide*.

## What helps ensure data accuracy and consistency?

DB2 provides many controls that can be applied to data entry and update. Some of the controls are automatic, some optional. All prohibit certain operations and provide error or warning messages if those operations are attempted. The following sections relate the operations to typical auditing concerns.

The set of techniques in this section is not exhaustive. Other combinations of techniques are possible; for example, you can use table check constraints or a view with the check option to ensure that data values are members of a certain set, rather than set up a master table and define referential constraints. In all cases, you can enforce the controls through application programs, and restrict the INSERT and UPDATE privileges only to those programs.

### Is required data present? Is it of the required type?

To ensure that required data is present, define columns with the NOT NULL clause.

The assignment of column data types and lengths also provides some control on the type of data. Alphabetic data cannot be entered into a column with one of the numeric data types, data entered into a DATE or TIME column must have an acceptable format, and so on.

For suggestions about assigning column data types and the NOT NULL attribute, see *DB2 SQL Reference*.

### Are data values unique where required?

The preferred control is to create a unique index on the column or set of columns in question. The same method completes the definition of a primary key for a table. See An Introduction to DB2 for OS/390 for suggestions about indexes.

### Has data a required pattern? Is it in a specific range?

Triggers and table check constraints enhance the ability to control data integrity.

Triggers are very powerful for defining and enforcing rules that involve different states of DB2 data. For example, a rule prevents a salary column from being increased by more than ten percent. A trigger can enforce this rule and provide the value of the salary before and after the increase for comparison. See Chapter 5 of *DB2 SQL Reference* for information using the CREATE TRIGGER statement to create a trigger.

A check constraint designates the values that specific columns of a base table can contain. Written in SQL, it can express not only simple constraints such as a required pattern or a specific range, but also rules that refer to other columns of the same table.

As an auditor, you might check that required constraints on column values are expressed as table check constraints in the table definition. For a full description of the rules for those constraints, see CREATE TABLE in Chapter 5 of *DB2 SQL Reference*.

**General-use Programming Interface**

An alternative technique is to create a view with the check option, and then insert or update values only through that view. For example, suppose that, in table T, data in column C1 must be a number between 10 and 20, and data in column C2 is an alphanumeric code that must begin with A or B. Create the view V1 with the following statement:

```
CREATE VIEW V1 AS
  SELECT * FROM T
    WHERE C1 BETWEEN 10 AND 20
    AND  (C2 LIKE 'A%' OR C2 LIKE 'B%')
WITH CHECK OPTION;
```

Only data that satisfies the WHERE clause can be entered through V1. See An Introduction to DB2 for OS/390 for information on creating and using views.

**End of General-use Programming Interface**

A view cannot be used with the LOAD utility, but that restriction does not apply to user-written exit routines. Several types of user-written routines are pertinent here:

**Validation routines** are expected to be used for validating data values. They access an entire row of data, can check the current plan name, and return a nonzero code to DB2 to indicate an invalid row.

**Edit routines** have the same access, and can also change the row that is to be inserted. They are typically used to encrypt data, substitute codes for lengthy fields, and the like; but they can also validate data and return nonzero codes.

**Field procedures** access data that is intended for a single column; they apply only to short-string columns. However, they accept input parameters, so generalized procedures are possible. A column that is defined with a field procedure can be compared only to another column that uses the same procedure.

See “Appendix B. Writing exit routines” on page 901 for information about using exit routines.

## Is new data in a specific set? Is it consistent with other tables?

These question are answered by referential integrity, a key feature of DB2. When you define primary and foreign keys, DB2 automatically enforces the rule that every value of a foreign key in a dependent table must be a value of the primary key of the appropriate parent table. For information about the means, implications, and limitations of enforcing referential integrity, see *DB2 Application Programming and SQL Guide*.

You can use this method to ensure that data in a column takes on only specific values. Set up a master table of allowable values and define its primary key. Define foreign keys in other tables that must have matching values in their columns; a delete rule of SET NULL is often appropriate.

DB2 does not enforce referential constraints across subsystems.

## What ensures that updates are tracked?

Triggers offer an efficient means of maintaining an audit trail. A triggering operation can be a DELETE, INSERT, or UPDATE that names the SQL data change operation for which the trigger is activated.

For example, you can qualify a trigger UPDATE operation by providing a list of column names. The trigger is only activated when one of the named columns is updated. A trigger that performs validation for changes that are made in an UPDATE operation must access column values both before and after the update. Transition variables (only available to row triggers) contain the column values of the affected row for which a trigger was activated. The old column values prior to the triggering operation and the new column values after the triggering operation are both available.

See *DB2 SQL Reference* for information about when to use triggers.

## What ensures that concurrent users access consistent data?

If you do not use uncommitted read (UR) isolation, DB2 automatically controls access using locks. You can trade locking resources among concurrent users, but you cannot violate the basic principle of locking control. No program can access data that another program changed but not yet committed.

However, if you use uncommitted read (UR) isolation, you can violate that basic principle of locking. Uncommitted read (UR) isolation lets users to see uncommitted data. Although the data is physically consistent, a number of logical inconsistencies can occur, or the data could be wrong. The question for auditors then becomes, “How can I tell what applications use UR isolation?” For static SQL, the question can be answered by querying the catalog.

Use the following query to determine which plans use UR isolation:

```
SELECT DISTINCT Y.PLNAME
   FROM SYSIBM.SYSPLAN X, SYSIBM.SYSSTMT Y
   WHERE (X.NAME = Y.PLNAME AND X.ISOLATION = 'U')
       OR Y.ISOLATION = 'U'
   ORDER BY Y.PLNAME;
```

Use the following query to determine which packages use UR isolation:

```
SELECT DISTINCT Y.COLLID, Y.NAME, Y.VERSION
   FROM SYSIBM.SYSPACKAGE X, SYSIBM.SYSPACKSTMT Y
   WHERE (X.LOCATION = Y.LOCATION AND
          X.LOCATION = ' '        AND
          X.COLLID   = Y.COLLID   AND
          X.NAME     = Y.NAME     AND
          X.VERSION  = Y.VERSION  AND
          X.ISOLATION = 'U')
       OR Y.ISOLATION = 'U'
   ORDER BY Y.COLLID, Y.NAME, Y.VERSION;
```

For dynamic SQL statements, turn on performance trace class 3.

***Consistency between systems:*** Where an application program writes data to both DB2 and IMS, or DB2 and CICS, the subsystems prevent concurrent use of data until the program declares a point of consistency. For a detailed description of how data is kept consistent between systems, see “Consistency with other systems” on page 359.

## Have any transactions been lost or left incomplete?

Database balancing is a technique that helps to warn of such an occurrence. An application program that uses database balancing asks, for each set of data, whether the opening balance and the control totals plus the processed transactions equal the closing balance and control totals.

DB2 has no automatic mechanism to calculate control totals and column balances and compare them with transaction counts and field totals. To use database balancing, you must design these calculations into the application program. For example, you can have the program maintain a control table that contains information to balance the control totals and field balances for update transactions against a user's view. The control table might contain these columns:

- View name
- Authorization ID
- Number of logical rows in the view (not the same as the number of physical rows in the table)
- Number of insert and update transactions
- Opening balances
- Totals of insert and update transaction amounts
- Relevant audit trail information such as date, time, terminal ID, and job name

The program updates the transaction counts and amounts in the control table each time it completes an insert or update to the view, and commits the work only after updating the control table, to maintain coordination during recovery. After processing all transactions, the application writes a report that verifies control total and balancing information.

## How can I tell that data is consistent?

Controlling data entry is not enough; you must also verify the results. The suggestions that follow can help to uncover errors or problems. Additionally, the DSN1CHKR utility verifies the integrity of the DB2 catalog and directory table spaces by scanning the specified table space for broken links, damaged hash chains, or orphan entries. For more information see Part 3 of *DB2 Utility Guide and Reference*.

### SQL queries

**General-use Programming Interface**

One relevant feature of DB2 is the ease of writing an SQL query to search for a specific type of error. For example, consider the view that is created on page 227; it is designed to allow an insert or update to table T1 only if the value in column C1 is between 10 and 20 and the value in C2 begins with A or B. To check that the control has not been bypassed, issue this statement:

```
SELECT * FROM T1
  WHERE  NOT (C1 BETWEEN 10 AND 20
  AND  (C2 LIKE 'A%' OR C2 LIKE 'B%'));
```

Ideally, no rows are returned.

You can also use SQL statements to get information from the DB2 catalog about referential constraints that exist. For several examples, see *DB2 SQL Reference*.

**End of General-use Programming Interface**

### Data modifications

Whenever an operation is performed that changes the contents of a data page or an index page, DB2 checks to verify that the modifications do not produce inconsistent data.

## CHECK utility

The CHECK utility also helps ensure data consistency in the following ways:

- CHECK INDEX checks the consistency of indexes with the data that the indexes must point to: Does each pointer point to a data row with the same value of the index key? Does each index key point to the correct LOB?
- CHECK DATA checks referential constraints: Is each foreign key value in each row actually a value of the primary key in the appropriate parent table?
- CHECK DATA checks table check constraints and checks the consistency between a base table space and its associated LOB table spaces: Is each value in a row within the range that was specified for that column when the table was created?
- CHECK LOB checks the consistency of a LOB table space: Are any LOBs in the LOB table space invalid?

See *DB2 Utility Guide and Reference* for more information on CHECK.

## DISPLAY DATABASE command

If a table is loaded without enforcing referential constraints on its foreign key columns, it can contain data that violates the constraints. The table space containing the table is placed in the check-pending status. You can determine which table spaces are in that status by using the DISPLAY DATABASE command with the RESTRICT option. You can also display table spaces with invalid LOBs. See Chapter 2 of *DB2 Command Reference* for information about using this command.

## REPORT utility

You might want to determine which table spaces contain a set of tables that are interconnected by referential constraints or which LOB table spaces are associated with which base tables. See *DB2 Utility Guide and Reference* for information about using the REPORT utility.

## Operation log

An operation log verifies that DB2 is operated reliably or reveals unauthorized operation and overrides. It consists of an automated log of DB2 operator commands (such as starting or stopping the subsystem or its databases) and any abend of DB2. The recorded information includes: command or condition type, date, time, authorization ID of the person issuing the command, and database condition code.

You can obtain this information from the system log (SYSLOG), the SMF data set, or the automated job scheduling system, using SMF reporting, job scheduler reporting, or a user-developed program. You should review the log report daily and keep a history file for comparison. Because abnormal DB2 termination can indicate integrity problems, an immediate notification procedure should be in place to alert the appropriate personnel (DBA, systems supervisor, and so on).

## Internal integrity reports

***For application programs:*** Standardized procedures should exist to record any DB2 return codes that are received that indicate possible data integrity problems—inconsistency between index and table information, physical errors on database disk, and so on. All programs must check the SQLCODE or the SQLSTATE for the return code that is issued after an SQL statement is run. DB2 records, on SMF, the occurrence (but not the cause) of physical disk errors and application program abends. The program can retrieve and reported this information; the system log (SYSLOG) and the DB2 job output listing also have this information. However, in some cases, only the program can provide enough detail to identify the exact nature of problem.

You can incorporate the standardized procedure into application programs or it can exist separately as part of an interface. The procedure records the incident in a history file and writes a message to the operator's console, a database administrator's TSO terminal, or a dedicated printer for certain codes. The recorded information includes the date, time, authorization ID, terminal ID or job name, application, view or table affected, error code, and error description. You should daily review reports by time and by authorization ID.

***For utilities:*** When a DB2 utility reorganizes or reconstructs data in the database, it produces statistics to verify record counts and report errors. The LOAD and REORG utilities produce data record counts and index counts to verify that no records were lost. In addition to that, keep a history log of any DB2 utility that updates data, particularly REPAIR. Regularly produce and review these reports, which you can obtain through SMF customized reporting or a user-developed program.

## How can DB2 recover data after failures?

DB2 provides extensive methods of recovering data after a subsystem, media, or program failure. If a subsystem fails, a restart of DB2 automatically restores the integrity of the data by backing out uncommitted changes and completing the processing of committed changes. If a media failure occurs (such as physical damage to a data storage device), the RECOVER utility can recover data to the current point. If a program error occurs, the RECOVER utility can recover data to a specific log record or to a specific image copy. For detailed information and recommendations, see “Recovering page sets and data sets” on page 393.

The recovery methods require adequate image copies of table spaces that are to be recovered and the integrity of the log data sets. A database administrator might need to develop and use queries against the SYSIBM.SYSCOPY table to verify that image copies were made appropriately. The REPORT utility can also provide some of that information.

The bootstrap data set (BSDS) maintains an inventory of all archive log data sets, including the time and date the log was created, the data set name, its status, and other information. The print log map utility can list the log data set inventory from the BSDS. Run and review the print log map utility daily ensure that archive data sets have been created properly and that they are readily available for use in recovery.

In the event that a program failure affects a COMMIT operation, a user-written program can read the DB2 log to determine exactly what change was made (except for LOB data in LOB table spaces that are LOG NO).

If IMS, CICS, or a remote DBMS is that is attached to DB2 when a failure occurs, DB2 coordinates restart with the other subsystem, keeping data consistent across all subsystems.

## How can I protect the software?

Whenever you install a new version, release, or maintenance of DB2, an automatic record provides an audit trail. The new release number is recorded by System Modification Program/Extended (SMP/E) when the DB2 subsystem programs and libraries are loaded. Each major component subsystem of DB2 has a function module identifier, which uniquely qualifies that subsystem to SMP/E. As part of the installation verification procedure, SMP/E records it in a history file along with a date and time, and can produce a report for management review. The audit trail aids in determining whether the changes are appropriate and whether they are made by authorized personnel and can also aid in investigation of application-related problems.

DB2 load modules need the same protection as those for any system program. For ways of protecting the system, refer to the appropriate MVS publication. The DB2 subsystem initialization load module (typically DSNZPARM) deserves special consideration, for it contains the IDs that hold the broad authorities of installation SYSADM and installation SYSOPR.

## How can I ensure efficient usage of resources?

The DB2 tools that can help you make efficient use of your resources are described in “Chapter 28. Improving resource utilization” on page 579. The following tools can be particularly useful:

- The resource limit facility (governor) limits the amount of time a dynamically issued query can use. The governor records these limits in a resource limit specification table (RLST). For details, see “Resource limit facility (governor)” on page 581 .
- The accounting trace is similar to the audit trace that is described in this chapter. Use it to collect start and stop times, numbers of commits, counts of the use of certain SQL statements, and CPU times. For details, see “DB2 trace” on page 1033.
- The performance trace is also pertinent. It can provide an enormous amount of detail, and is usually used for investigating particular problems. For more information, see “DB2 trace” on page 1033.

# Chapter 15. A sample security plan for employee data

This chapter shows one approach to enforcing a security plan by using authorization IDs, implicit privileges, granted privileges and authorities, and the audit trace. For example, suppose that the sample enterprise, the Spiffy Computer Company, decides on a list of objectives for the security of employee data. The list is a compromise between two basic motivations:

- Employees should not be able to browse the employee table to find out the salary, bonus, or commission that is paid to other employees. They definitely should not be able to update those values, for themselves or others.
- Managers have legitimate reasons for knowing the compensations that are paid to people who report to them. And someone must be able to make changes to salary data.

The Spiffy management derives the detailed list of objectives that follows. Do not view it as a model security plan; it is only a sample, chosen to illustrate various possibilities and expose certain problem areas. Your own security plans will be different.

***The security objectives:*** The security objectives for Spiffy's security plan are:

- Managers can see, but not update, all the employee data for members of their own departments. Managers of managers can see all the data for employees of departments under them.
- The employee table resides at a central location. Managers at remote locations can query the data in that table.
- Changes to the employee table are made by a Payroll Operations department. (It is not listed in the sample department table.) Department members can update any column of the employee table except for salary, bonus, and commission, and any row except those for members of their own department. Changes to the table are made only from the central location; hence, payroll operations are not affected by distributed access.
- Changes to salary, bonus, and commission amounts are made through another table. The table lists an employee ID and a salary update, for example; the row can be inserted by a member of Payroll Operations. When a list of changes is complete, it must be verified by another group, Payroll Management, who can then transfer the changes to the employee table.
- No one else can see the employee data. (This objective cannot actually be fully achieved. At the very least, some ID must occasionally exercise powers that are reserved to SYSADM authority, and at that time that ID can retrieve any data in the system. The security plan uses the trace facility to monitor the use of that power.)

## Managers' access

Managers can retrieve, but not change, all information in the employee table for members of their own departments. Managers of managers have the same privileges for their own departments and the departments immediately under them. Those restrictions can most easily be implemented by views.

For example, you can create a view of employee data for every employee reporting to a manager—even if more than one department are involved. Such a view requires altering department table DSN8410.DEPT by adding a column to contain managers' IDs:

```
ALTER TABLE DSN8710.DEPT
   ADD MGRID CHAR(8) FOR SBCS DATA NOT NULL WITH DEFAULT;
```

Every manager should have the SELECT privilege on a view that is created as follows:

```
CREATE VIEW DEPTMGR AS
   SELECT * FROM DSN8710.EMP, DSN8710.DEPT
      WHERE WORKDEPT = DEPTNO
      AND MGRID = USER;
```

## To what ID is the SELECT privilege granted?

Assuming that nearly every employee of the Spiffy Computer Company has a TSO logon ID and a password, and can access DB2I or QMF. The security planners can take one of two approaches to granting privileges:

- Grant privileges to individual IDs and revoke them if the user of the ID leaves the company or transfers to another position. This is called the *individual* approach.
- Create RACF groups and grant privileges to the group IDs, with the intention of never revoking them. When a individual ID needs those privileges, connect it to the group; disconnect it when its user leaves or transfers. This is called the *functional* approach. Another example of grouping is when many authorization IDs are translated into a single outbound ID.

The functional approach is probably more convenient in the following situations:

- Many different privileges are required, and when they are revoked from one individual, they must be granted to another. In that case, the set of privileges probably constitutes a function of the enterprise, which must persist even though the individual now performing it leaves or transfers.
- Several users need the same set of privileges. Again, the set probably constitutes a business function.
- The privileges are given with the grant option, or they let users create objects that must persist after their original owners leave or transfer. In both cases, revoking the privileges might not be appropriate. The revokes cascade to other users, and to change ownership, you might need to drop objects and re-create them.

What about the managers' views of their own departments? In theory, the privilege of selecting from the view is part of the function of managing. If a manager transfers, another is appointed. That suggests the functional approach.

However, in this case, one privilege is needed—SELECT on a particular view. The privilege does not carry the grant option, and it does not allow creating new objects. So the individual approach might be just as convenient.

Actually, Spiffy Computer does not need to make a permanent choice immediately. Which approach to use is a matter of convenience; where both produce the same results, either can be used. Both approaches can also be used simultaneously; some departments could be represented by their managers' individual IDs, others could be represented by group IDs, and the company could change gradually from one approach to the other.

So the security plan starts out by using the individual approach, with the intent of re-examining the system later. Initially, all managers are given the SELECT privilege on the views for their departments by statements like this one:

```
GRANT SELECT ON DEPTMGR TO EMP0060;
```

That assumes that EMP0060 is the individual ID of employee 000060, who is the manager of one or more departments.

## Allowing distributed access

The security plan envisions that managers at remote locations will query data in the employee table at a central, serving location. The restrictions on the data they are allowed to query can most easily be implemented by views, just like the view for managers who use the central DB2 location directly. The remaining questions are:

- What IDs should have privileges on those views?
- How is responsibility for those IDs divided between the central location and the remote locations?

Spiffy's security plan answers those questions as follows. Again, this plan is not a recommendation for your own security needs—it is merely an example of what is possible.

- Privileges on views for departments at remote locations are given to IDs that are managed at the central location. For example, the ID MGRD11 has the SELECT privilege on the view DEPTD11.
- If the manager of Department D11 uses a remote system, the ID there must be translated to MGRD11 before a request is sent to the central system. All other IDs are translated to CLERK before they are sent to the central system.
- The translated IDs, like MGRD11, are managed through the communications database.
- An ID from a remote system must be authenticated on any request to the central system.

The means of implementing these decisions are described in the following sections:

“Actions at the central server location” and
“Actions at remote locations” on page 236.

### Actions at the central server location

To implement the provisions of the security plan, the central DB2 system must take the following actions:

- Authenticate every incoming ID with RACF.
- For SNA connections, provide an entry in table SYSIBM.LUNAMES, in the CDB, for the LUNAME of every remote location. The entry must specify that connections must be verified. Table 57 shows what one such entry might look like.

*Table 57. The SYSIBM.LUNAMES table at the central location*

| LUNAME | USERNAMES | SECURITY_IN | ENCRYPTPSWDS |
|---|---|---|---|
| LUREMOTE | blank | V | N |

(The security plan treats all remote locations alike, so it does not require encrypting passwords. That option is available only between two DB2 subsystems that use SNA connections.)

- For TCP/IP connections, make sure the TCP/IP ALREADY VERIFIED field of installation panel DSNTIP5 is NO. This ensures that incoming requests that use TCP/IP are not accepted without authentication.
- Grant all privileges and authorities that are required by the manager of Department D11 to the ID MGRD11.

### Actions at remote locations

To implement the provisions of the security plan, a remote DB2 subsystem must take the actions described below. (For a system other than DB2 for OS/390 and z/OS, the actions might be somewhat different; check the documentation for the product you are using. The remote system must satisfy the requirements that are already imposed by the central system.)

- For SNA connections, provide an entry in table SYSIBM.LUNAMES for the LUNAME of the central location. The entry must specify outbound ID translation for attachment requests to that location. Table 58 shows what such an entry might look like.

*Table 58. The SYSIBM.LUNAMES table at the remote location*

| LUNAME | USERNAMES | SECURITY_OUT |
|---|---|---|
| LUCENTRAL | O | R |

- For TCP/IP connections, provide an entry in table SYSIBM.IPNAMES for the LUNAME that is used by the central location. (The LUNAME is used to generate RACF PassTickets.) The entry must specify outbound ID translation for requests to that location. Table 59 shows what such an entry might look like.

*Table 59. The SYSIBM.IPNAMES table at the remote location*

| LINKNAME | USERNAMES | SECURITY_OUT | IPADDR |
|---|---|---|---|
| LUCENTRAL | O | R | central.vnet.ibm.com |

- Provide entries in table SYSIBM.USERNAMES to translate outbound IDs. In this example, MEL1234 is translated to MGRD11 before it is sent to the LU name that is specified in the LINKNAME column. All other IDs are translated to CLERK before they are sent to that LU. Table 60 shows what such an entry might look like.

*Table 60. The SYSIBM.USERNAMES table at the remote location*

| TYPE | AUTHID | LINKNAME | NEWAUTHID |
|---|---|---|---|
| O | MEL1234 | LUCENTRAL | MGRD11 |
| O | blank | LUCENTRAL | CLERK |

## Auditing managers’ use

The payroll data is extremely sensitive; therefore, the security plan calls for automatically starting the audit trace for all classes whenever DB2 is started. The employee table is to be created with AUDIT ALL, so an audit record exists for every access to the table. Every week, the records are scanned to report the number of accesses by each manager.

The report highlights any number outside an expected range. The system operator makes a summary of the reports every two months, and scans it for unusual patterns of access. A large number of accesses or an unusual pattern might reveal use of a manager's logon ID by another, unauthorized employee.

## Payroll operations

To satisfy the stated security objectives for members of Payroll Operations, the security plan again uses a view. The view shows all the columns of the table *except* those for job, salary, bonus, and commission; the view also shows all rows *except* those for members of the Payroll Operations department. Members of Payroll

Operations have SELECT, INSERT, UPDATE, and DELETE privileges on the view; and the privileges are granted WITH CHECK OPTION, so that they cannot insert values that exceed the limits of the view.

A second, similar view gives Payroll Management the privilege of retrieving and updating any record, including those of Payroll Operations. Neither view, though, allows updates of compensation amounts. When a row is inserted for a new employee, the compensation amounts are left null, to be changed later by an update.

Both views are created and owned by, and privileges are granted by, the owner of the employee table.

## Salary updates

The plan does not allow members of Payroll Operations to update compensation amounts directly. Instead, another table exists, the “payroll update table”, containing only the employee ID, job, salary, bonus, and commission. Members of Payroll Operations make all job, salary, and bonus changes to the payroll update table, except those for their own department. After the prospective changes are verified, the manager of Payroll Operations runs an application program that reads the payroll update table and makes the corresponding changes to the employee table. Only that program, the “payroll update program”, has the privilege of updating job, salary, and bonus in the employee table.

Calculating commission amounts at Spiffy Computer Company are handled separately. Commissions are calculated by a complicated arithmetic formula that considers the employee's job, department, years of service with the company, and responsibilities for various projects and project activities. The formula is embodied in an application plan, the “commission program”, which is run regularly to insert new commission amounts in the payroll update table. The plan owner must have the SELECT privilege on the employee table and other tables.

## Additional controls

The separation of potential salary changes into the payroll update table allows them to be verified before they go into effect; at Spiffy Computer Company, the changes are checked against a written change request that is signed by a required level of management. That is considered the most important control on salary updates, but the plan also includes these other controls:

- The employee ID in the payroll update table is a foreign key column that refers to the employee ID in the employee table. Enforcing the referential constraint prevents assigning a change to an invalid employee ID.
- The employee ID in the payroll update table is also a primary key for that table, so its values are unique. Because of that, in any one operating period (such as a week) all the changes for any one employee must appear in the same row of the table. No two rows can carry conflicting changes.
- The plan documents an allowable range of salaries, bonuses, and commissions for each job level. The security planners considered the following ways to ensure that updates would stay within those ranges:
  - Keep the ranges in a DB2 table and, as one step in verifying the updates, query the payroll update table and the table of ranges, retrieving any rows for which the planned update is outside the allowed range.
  - Build the ranges into a validation routine, and apply it to the payroll update table to automatically reject any insert or update that is outside its allowed range.

- – Embody the ranges in a view of the payroll table, using WITH CHECK OPTION, and make all updates to the view. The ID that owns the employee table also owns the view.
- – Create a trigger to prevent salaries, bonuses, and commissions from being increased by more than the percent allowed for each job level. See *DB2 SQL Reference* for more information about using triggers.
- – Create the table with table check constraints for the salaries, bonuses, and commissions. The planners chose this approach because it is both simple and easy to control. See Part 1 of *DB2 Application Programming and SQL Guide* for information about using table check constraints.

## To what ID are privileges granted?

The plan for the Payroll Operations department strongly suggests the functional approach, for these reasons:

- Several privileges are needed—the privileges on the views and probably also the EXECUTE privilege on the application plan for the commission program.
- Several members of the department must all have the same set of privileges.
- If members of the department leave, others are hired or transfer in.

Therefore, the security plan calls for creating a RACF group for Payroll Operations. All required privileges are granted to the group ID, with the intent not to revoke them. The primary IDs of new members of the department are connected to the group ID, which becomes a secondary ID for each of them. The primary IDs of members who leave the department are disconnected from the group.

DB2USER can define the group, as described in “Add RACF groups” on page 206. DB2USER could retain ownership of the group, or it could assign the ownership to an ID that is used by Payroll Management. The privileges that the group needs can be granted by the owner of the employee table.

## Auditing use by payroll operations and payroll management

Like the employee table, the payroll update table is created with AUDIT ALL. For both tables, the numbers of accesses by the payroll operations and payroll management groups are reported. A summary of accesses of the employee table by the payroll update program is also reported. Like the reports of managers’ accesses, the reports of payroll accesses are scanned for large numbers or unusual patterns of access.

# Others who have access

In addition to the privileges of managers, and of members of the Payroll Operations and Payroll Management groups, the security plan considers the privileges of database administrators, system administrators, and owners of tables, views, packages, and application plans.

## IDs with database administrative authority

An ID with DBADM authority over database DSN8D71A, which holds the employee table, can select from, insert into, delete from, update, or alter any table in the database, and create and drop indexes on the tables. The security planners did not need to grant that authority to any ID. Regular operations require no more than an ID with DBCTRL authority. That ID could copy tables, recover any table space, run the CHECK utility, and generally support the continued availability of the database without actually being able to retrieve or change the data.

However, database DSN8D71A contains several other tables (which are all described in “Appendix A. DB2 sample tables” on page 883). The planners considered putting the payroll tables into another database. That way, those with access to DSN8D71A could not access them.

Planners decided to have an administrative ID that could access those fully, functional approach to privileges. Although the authorities that DB2 provides, like DBADM, are convenient collections of privileges for many purposes, they are not the only collections that can be needed. The security plan called for a RACF group that had:

1. DBCTRL authority over DSN8D71A
2. The INDEX privilege on all tables in the database except the employee and payroll update tables
3. The SELECT, INSERT, UPDATE, and DELETE privileges on selected tables

The privileges are to be granted to the group ID by an ID with SYSADM authority.

## IDs with system administrative authority

An ID with SYSADM authority can access sensitive data that is not only in the employee and payroll update tables, but also in any other table in the entire DB2 subsystem. However, that authority can be needed only intermittently and for relatively short periods.

Because such sweeping authority must be controlled at the highest level, the security plan calls for giving it to DB2OWNER, the ID that is responsible for DB2 security. That does not mean that only IDs that are connected to DB2OWNER can exercise all that authority, grant privileges on every plan and package,and initiate every use of the STOSPACE utility. Instead, DB2OWNER can grant privileges to a group, connect other IDs to the group as needed, and later disconnect them.

Also, DB2OWNER can grant SYSCTRL authority to selected IDs. IDs with SYSCTRL authority can exercise most of the privileges of SYSADM authority and can assume much of the day-to-day work. Those IDs cannot access data directly or run plans, unless the privileges for those actions are explicitly granted to them; but they can run utilities and examine the output data sets, or grant privileges that would allow other IDs to access data. Thus, accessing sensitive data is somewhat inconvenient, but not impossible.

Grants of the BINDAGENT privilege can also relieve the need to have SYSADM authority continuously available. IDs with the BINDAGENT privilege can bind plans and packages on behalf of another ID, but they cannot run the plans they bind without being explicitly granted the EXECUTE privilege.

## The employee table owner

Spiffy Computer Company can never fully achieve its stated objective that only a manager can retrieve an employee’s data record. In planning the necessary views and GRANT statements, the security planners must consider the ID that owns the views and grants the privileges. That ID implicitly has the SELECT privilege on the employee table.

The activities that are planned for the Payroll Operations and Payroll Management departments require a new table and several new views. The security plan calls for all of those to be owned by the owner of the employee table.

The planned activities also use these programs, whose owners must also have certain privileges.

- The owner of the payroll update program must have the SELECT privilege on the payroll update table and the UPDATE privilege on the employee table.
- The owner of the commission program must have the UPDATE privilege on the payroll update table and the SELECT privilege on the employee table.
- Several other payroll programs do the usual payroll processing—printing payroll checks, writing summary reports, and so on.

At this point, the security planners adopt an additional objective for the plan: to limit the number of IDs that have any privileges on the employee table or the payroll update table to the smallest convenient value. To meet that objective, they decide that all the CREATE VIEW and GRANT statements are to be issued by the owner of the employee table. Hence, the security plan for employee data assigns several key activities to that ID. The security plan considers the need to:

- Revoke and grant the SELECT privilege on a manager's view whenever a department's manager is changed
- Drop and create managers' views whenever a reorganization of responsibilities changes the list of department identifiers
- Maintain the view through which the employee table is updated

The privileges for those activities are implicit in ownership of the employee table and the views on it. The same ID must also:

- Own the application plans and packages for the payroll program, the payroll update program, and the commission program
- Occasionally acquire ownership of new application plans and packages

For those activities, the ID requires the BIND or BINDADD privileges. For example, an ID in Payroll Management can, through the SELECT privilege on the employee table, write an SQL query to retrieve average salaries by department, for all departments. To create an application plan that contains the query requires the BINDADD privilege.

Again, the list of privileges suggests the functional approach. The owner of the employee table is to be a RACF group ID.

## Auditing for other users

Any access to the employee or payroll update tables by anyone other than the department managers, the Payroll Operations and Payroll Management groups, and the payroll update program, is considered an exception. Those exceptions are listed in full, and each is checked to see that it was a planned operation by the users with SYSADM or DBADM authority, or the tables' owner.

Denials of access to the table are also listed. Those represent attempts by unauthorized IDs to use the tables. Some are possibly accidental; others can be attempts to break the security system.

After running the periodic reports, the audit records are archived. They provide a complete audit trail of access to the employee data through DB2.

# Part 4. Operation and recovery

# Chapter 16. Basic operation

The information under this heading, up to “Running IMS application programs” on page 260, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

The simplest elements of operation for DB2 for OS/390 and z/OS are described in this chapter; they include:
“Entering commands”
“Starting and stopping DB2” on page 256
“Submitting work to be processed” on page 259
“Receiving messages” on page 263

Normal operation also requires more complex tasks. They are described in:

- “Chapter 17. Monitoring and controlling DB2 and its connections” on page 267, which considers the control of connections to IRLM, to TSO, to IMS, and to CICS, as well as connections to other database management systems.
- “Chapter 18. Managing the log and the bootstrap data set” on page 331, which describes the roles of the log and the bootstrap data set in preparing for restart and recovery.
- “Chapter 19. Restarting DB2 after termination” on page 347, which tells what happens when DB2 terminates normally or abnormally and how to restart it while maintaining data integrity.
- “Chapter 20. Maintaining consistency across multiple systems” on page 359, which explains the two-phase commit process and the resolution of indoubt units of recovery.
- “Chapter 21. Backing up and recovering databases” on page 373, which explains how to prepare for recovery as well as how to recover.

Recovery after various types of failure is described in:

- “Chapter 22. Recovery scenarios” on page 409
- “Chapter 23. Recovery from BSDS or log failure during restart” on page 475

***Operating a data sharing group:*** Although many of the commands and operational procedures described here are the same in a data sharing environment, some special considerations are described in Chapter 5 of *DB2 Data Sharing: Planning and Administration*. In particular, consider the following issues when you are operating a data sharing group:

- New commands used for data sharing, and the concept of command scope
- Logging and recovery operations
- Restart after an abnormal termination
- Disaster recovery procedures
- Recovery procedures for coupling facility resources

## Entering commands

You can control most of the operational environment by using DB2 commands. You might need to use other types of commands, including:

- IMS commands that control IMS connections
- CICS commands that control CICS connections
- IMS and CICS commands that allow you to start and stop connections to DB2 and display activity on the connections

- MVS commands that allow you to start, stop, and change the internal resource lock manager (IRLM)

Using these commands is described in “Chapter 17. Monitoring and controlling DB2 and its connections” on page 267. For a full description of the commands available, see Chapter 2 of *DB2 Command Reference*.

## DB2 operator commands

The DB2 commands, as well as their functions, are:

**ALTER BUFFERPOOL**
Sets or alters buffer pool size while DB2 is online.

**ALTER GROUPBUFFERPOOL**
Alters attributes of group buffer pools, which are used in a data sharing environment.

**ALTER UTILITY**
Changes parameter values of the REORG utility while REORG is running.

**ARCHIVE LOG**
Archives (offloads) the current active log.

**CANCEL THREAD**
Cancels processing for specific local or distributed threads. It can be used for parallel task threads.

**DISPLAY ARCHIVE**
Displays information about the specifications for archive parameters, status of allocated dedicated tape units, volume and data set names associated with all active tape units, and correlation ID of the requester.

**DISPLAY BUFFERPOOL**
Displays buffer pool information while DB2 is online.

**DISPLAY DATABASE**
Displays the status of a database.

**DISPLAY DDF**
Displays information about the status and configuration of the distributed data facility (DDF), and about the connections or threads controlled by DDF.

**DISPLAY FUNCTION SPECIFIC**
Displays the statistics about external user-defined functions accessed by DB2 applications.

**DISPLAY GROUP**
Displays information about the data sharing group to which a DB2 subsystem belongs.

**DISPLAY GROUPBUFFERPOOL**
Displays status and statistical information about DB2 group buffer pools, which are used in a data sharing environment.

**DISPLAY LOCATION**
Displays statistics about threads and conversations between remote DB2 subsystem and the local subsystem.

**DISPLAY LOG**
Displays the current checkpoint frequency (CHKFREQ) value, information about the current active log data sets, and the status of the offload task.

**DISPLAY PROCEDURE**
Displays statistics about stored procedures accessed by DB2 applications.

**DISPLAY RLIMIT**
Displays the status of the resource limit facility (governor).

**DISPLAY THREAD**
Displays information about DB2, distributed subsystem connections, and parallel tasks.

**DISPLAY TRACE**
Displays the status of DB2 traces.

**DISPLAY UTILITY**
Displays the status of a utility.

**MODIFY TRACE**
Changes the trace events (IFCIDs) being traced for a specified active trace.

**RECOVER BSDS**
Reestablishes dual bootstrap data sets.

**RECOVER INDOUBT**
Recovers threads left indoubt after DB2 is restarted.

**RECOVER POSTPONED**
Completes backout processing for units of recovery (URs) whose backout was postponed during an earlier restart, or cancels backout processing of the postponed URs if the CANCEL option is used.

**RESET INDOUBT**
Purges DB2 information about indoubt threads.

**SET ARCHIVE**
Controls or sets the limits for the allocation and the deallocation time of the tape units for archive log processing.

**SET LOG**
Modifies the checkpoint frequency (CHKFREQ) value dynamically without changing the value in the subsystem parameter load module.

**SET SYSPARM**
Loads the subsystem parameter module specified in the command.

**START DATABASE**
Starts a list of databases or table spaces and index spaces.

**START DB2**
Initializes the DB2 subsystem.

**START DDF**
Starts the distributed data facility

**START FUNCTION SPECIFIC**
Activates an external function that is stopped.

**START PROCEDURE**
Starts a stored procedure that is stopped.

**START RLIMIT**
Starts the resource limit facility (governor).

**START TRACE**
Starts DB2 traces.

**STOP DATABASE**
Stops a list of databases or table spaces and index spaces.

**STOP DB2**
Stops the DB2 subsystem.

**STOP DDF**
Stops or suspends the distributed data facility.

**STOP FUNCTION SPECIFIC**
Prevents DB2 from accepting SQL statements with invocations of the specified functions.

**STOP PROCEDURE**
Prevents DB2 from accepting SQL CALL statements for a stored procedure.

**STOP RLIMIT**
Stops the resource limit facility (governor).

**STOP TRACE**
Stops traces.

**TERM UTILITY**
Terminates execution of a utility.

## Where DB2 commands are entered

You can enter commands from the following sources:

- An MVS console or MVS application program
- An IMS terminal or program
- A CICS terminal
- A TSO terminal
- An APF-authorized program
- An IFI application program

***From an MVS console or MVS application program:*** You can enter all DB2 commands from an MVS console or MVS application program. The START DB2 command can be entered only from the MVS console. The command group authorization level must be SYS.

More than one DB2 subsystem can run under MVS. You prefix a DB2 command with special characters that identify which subsystem to direct the command to. The 1- to 8-character prefix is called the *command prefix*. Specify the command prefix on installation panel DSNTIPM. The default character for the command prefix is -DSN1. Most examples in this book use the old default, the hyphen (-).

***From an IMS terminal or program:*** You can enter all DB2 commands except -START DB2 from either an IMS terminal or program. The terminal or program must be authorized to enter the /SSR command.

An IMS subsystem can attach to more than one DB2 subsystem, so you must prefix a command that is directed from IMS to DB2 with a special character that tells which subsystem to direct the command to. That character is called the *command recognition character* (CRC); specify it when you define DB2 to IMS, in the subsystem member entry in IMS.PROCLIB. (For details, see Part 2 of *DB2 Installation Guide*.)

If it is possible in your configuration, it can be less confusing if you make the CRC and the command prefix the same character for the same DB2 subsystem. If you are using a command prefix of more than one character, this is not possible.

The examples in this book assume that both the command prefix and the CRC are the hyphen (-). But if you can attach to more than one DB2 subsystem, you must prefix your commands with the appropriate CRC. In the following example, the CRC is a question mark character:

You enter:

```
/SSR ?DISPLAY THREAD
```

and DB2 returns the following messages:

```
DFS058  SSR COMMAND COMPLETED
DSNV401I ? DISPLAY THREAD REPORT FOLLOWS -
DSNV402I ? ACTIVE THREADS -
⋮
```

***From a CICS terminal:*** You can enter all DB2 commands except START DB2 from a CICS terminal authorized to enter the DSNC transaction code.

For example, you enter:

```
DSNC -DISPLAY THREAD
```

and DB2 returns the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
⋮
```

CICS can attach to only one DB2 subsystem at a time; therefore CICS does not use the DB2 command prefix. Instead, each command entered through the CICS attachment facility must be preceded by a hyphen (-), as in the example above. The CICS attachment facility routes the commands to the connected DB2 subsystem and obtains the command responses.

***From a TSO terminal:*** You can enter all DB2 commands except -START DB2 from a DSN session.

For example, the system displays:

```
READY
```

You enter:

```
DSN SYSTEM (subsystem-name)
```

The system displays:

```
DSN
```

You enter:

```
-DISPLAY THREAD
```

and DB2 returns the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
⋮
```

A TSO session can attach to only one DB2 subsystem at a time; therefore TSO does not use the DB2 command prefix. Instead, each command entered through the TSO attachment facility must be preceded by a hyphen (-), as in the example above. The TSO attachment facility routes the command to DB2 and obtains the command response.

All DB2 commands except START DB2 can also be entered from a DB2I panel using option 7, *DB2 Commands*. For more information on using DB2I, see “Using DB2I (DB2 Interactive)” on page 259.

***From an APF-authorized program:*** As with IMS, DB2 commands can be passed from an APF-authorized program to multiple DB2 subsystems by the MGCR (SVC 34) MVS service. Thus, the value of the command prefix identifies the particular subsystem to which the command is directed. The subsystem command prefix is specified, as in IMS, when DB2 is installed (in the SYS1.PARMLIB member IEFSSNxx). DB2 supports the MVS WTO Command And Response Token (CART) to route individual DB2 command response messages back to the invoking application program. Use of the CART token is necessary if multiple DB2 commands are issued from a single application program.

For example, to issue DISPLAY THREAD to the default DB2 subsystem from an APF-authorized program run as a batch job, code:

```
MODESUPV DS    0H
         MODESET MODE=SUP,KEY=ZERO
SVC34    SR    0,0
         MGCR  CMDPARM
         EJECT
CMDPARM  DS    0F
CMDFLG1  DC    X'00'
CMDLENG  DC    AL1(CMDEND-CMDPARM)
CMDFLG2  DC    X'0000'
CMDDATA  DC    C'-DISPLAY THREAD'
CMDEND   DS    0C
```

and DB2 returns the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
⋮

DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

***From an IFI application program:*** An application program can issue DB2 commands using the instrumentation facility interface (IFI). The IFI application program protocols are available through the IMS, CICS, TSO, and call attachment facility (CAF) attaches, and the Recoverable Resource Manager Services attachment facility. For an example in which the DB2 START TRACE command for monitor class 1 is issued, see “COMMAND: Syntax and usage” on page 1000.

## Where command responses go

In most cases, DB2 command responses are returned to the entering terminal or, for batch jobs, appear in the printed listing.

In CICS, you can direct command responses to another terminal. Name the other terminal as the destination (*dest*) in this command:

```
DSNC dest -START DATABASE
```

If a DB2 command is entered from an IMS or CICS terminal, the response messages can be directed to different terminals. If the response includes more than one message, the following cases are possible:

- If the messages are issued in a set, the entire set of messages is sent to the IMS or CICS terminal that entered the command. For example, DISPLAY THREAD issues a set of messages.
- If the messages are issued one after another, and not in a set, only the first message is sent to the terminal that entered the command. Later messages are routed to one or more MVS consoles via the WTO function. For example, START DATABASE issues several messages one after another.

  You can choose alternate consoles to receive the subsequent messages by assigning them the routing codes placed in the DSNZP*xxx* module when DB2 is installed. If you want to have all of the messages available to the person who sent the command, route the output to a console near the IMS or CICS master terminal.

For APF-authorized programs that run in batch jobs, command responses are returned to the master console and to the system log if hard copy logging is available. Hard copy logging is controlled by the MVS system command VARY. See *OS/390 MVS System Commands* for more information.

## Authorities for DB2 commands

The ability to issue DB2 commands, such as STOP DB2, and to use most other DB2 functions, requires the appropriate privilege or authority. Privileges and authorities can be granted to authorization IDs in many combinations and can also be revoked.

The individual authorities are listed in Figure 8 on page 109. Each administrative authority has the individual authorities shown in its box, and the individual authorities for all the levels beneath it. For example, DBADM has ALTER, DELETE, INDEX, INSERT, SELECT, and UPDATE authorities as well as those listed for DBCTRL and DBMAINT.

Any user with the STOPALL privilege can issue the STOP DB2 command. Besides those who have been granted STOPALL explicitly, the privilege belongs implicitly to anyone with SYSOPR authority or higher. When installing DB2, you can choose:
- One or two authorization IDs with installation SYSADM authority
- Zero, one, or two authorization IDs with installation SYSOPR authority

The IDs with those authorizations are contained in the load module for subsystem parameters (DSNZP*xxx*).

The START DB2 command can be entered only at an MVS console authorized to enter MVS system commands. The command group authorization level must be SYS.

DB2 commands entered from an MVS console are not associated with any secondary authorization IDs. The authorization ID associated with an MVS console is SYSOPR, which carries the authority to issue all DB2 commands except:
- RECOVER BSDS
- START DATABASE
- STOP DATABASE
- ARCHIVE LOG

APF-authorized programs that issue commands via MGCR (SVC 34) have SYSOPR authority. The authority to start or stop any particular database must be specifically granted to an ID with SYSOPR authority. Likewise, an ID with SYSOPR authority must be granted specific authority to issue the RECOVER BSDS and ARCHIVE LOG commands.

The SQL GRANT statement can be used to grant SYSOPR authority to other user IDs such as the /SIGN user ID or the LTERM of the IMS master terminal.

For information about other DB2 authorization levels, see “Establishing RACF protection for DB2” on page 198. *DB2 Command Reference* also has authorization level information for specific commands.

## Starting and stopping DB2

Starting and stopping DB2 is a simple process, and one that you will probably not have to do often. Before DB2 is stopped, the system takes a shutdown checkpoint. This checkpoint and the recovery log give DB2 the information it needs to restart.

This section describes the START DB2 and STOP DB2 commands, explains how you can limit access to data at startup, and contains a brief overview of startup after an abend.

## Starting DB2

When installed, DB2 is defined as a formal MVS subsystem. Afterward, the following message appears during any IPL of MVS:

```
DSN3100I - DSN3UR00 - SUBSYSTEM ssnm READY FOR -START COMMAND
```

where *ssnm* is the DB2 subsystem name. At that point, you can start DB2 from an MVS console that has been authorized to issue system control commands (MVS command group SYS), by entering the command START DB2. The command must be entered from the authorized console and not submitted through JES or TSO.

It is *not* possible to start DB2 by a JES batch job or an MVS START command. The attempt is likely to start an address space for DB2 that then abends, probably with reason code X'00E8000F'.

You can also start DB2 from an APF-authorized program by passing a START DB2 command to the MGCR (SVC 34) MVS service.

### Messages at start

The system responds with some or all of the following messages depending on which parameters you chose:

```
$HASP373 xxxxMSTR STARTED
DSNZ002I - SUBSYS ssnm SYSTEM PARAMETERS
           LOAD MODULE NAME IS dsnzparm-name
DSNY001I - SUBSYSTEM STARTING
DSNJ127I - SYSTEM TIMESTAMP FOR BSDS=87.267 14:24:30.6
DSNJ001I - csect CURRENT COPY n ACTIVE LOG DATA
           SET IS DSNAME=...,
           STARTRBA=...,ENDRBA=...
DSNJ099I - LOG RECORDING TO COMMENCE WITH
           STARTRBA = xxxxxxxxxxxx
$HASP373 xxxxDBM1 STARTED
DSNR001I - RESTART INITIATED
DSNR003I - RESTART...PRIOR CHECKPOINT RBA=xxxxxxxxxxxx
DSNR004I - RESTART...UR STATUS COUNTS...
           IN COMMIT=nnnn, INDOUBT=nnnn, INFLIGHT=nnnn,
           IN ABORT=nnnn, POSTPONED ABORT=nnnn
DSNR005I - RESTART...COUNTS AFTER FORWARD RECOVERY
```

```
          IN COMMIT=nnnn, INDOUBT=nnnn
DSNR006I - RESTART...COUNTS AFTER BACKWARD RECOVERY
          INFLIGHT=nnnn, IN ABORT=nnnn, POSTPONED ABORT=nnnn
DSNR002I - RESTART COMPLETED
DSN9002I - DSNYASCP 'START DB2' NORMAL COMPLETION
DSNV434I - DSNVRP NO POSTPONED ABORT THREADS FOUND
DSN9022I - DSNVRP 'RECOVER POSTPONED' NORMAL COMPLETION
```

If any of the *nnnn* values in message DSNR004I are not zero, message DSNR007I is issued to provide the restart status table.

The START DB2 command starts the system services address space, the database services address space, and, depending upon specifications in the load module for subsystem parameters (DSNZPARM by default), the distributed data facility address space and the DB2-established stored procedures address space. Optionally, another address space, the internal resource lock manager (IRLM), can be started automatically.

## Options at start

Starting invokes the load module for subsystem parameters. This load module contains information specified when DB2 was installed. For example, the module contains the name of the IRLM to connect to. In addition, it indicates whether the distributed data facility (DDF) is available and, if it is, whether it should be automatically started when DB2 is started. For information about using a command to start DDF, see “Starting DDF” on page 308. You can specify PARM (*module-name*) on the START DB2 command to provide a parameter module other than the one specified at installation.

There is a conditional restart operation, but there are no parameters to indicate normal or conditional restart on the START DB2 command. For information on conditional restart, see “Restarting with conditions” on page 355.

## Restricting access to data

You can restrict access to data with another option of the START DB2 command. Use:

**ACCESS(MAINT)**
To limit access to users who have installation SYSADM or installation SYSOPR authority.

Users with those authorities can do maintenance operations such as recovering a database or taking image copies. To restore access to all users, stop DB2 and then restart it. Either omit the ACCESS keyword or use:

**ACCESS(*)**
To allow all authorized users to connect to DB2.

## Wait state at start

If a JCL error, such as device allocation or region size, occurs while trying to start the database services address space, DB2 goes into wait status. To end the wait, cancel the system services address space and the distributed data facility address space from the console. After DB2 stops, check the start procedures of all three DB2 address spaces for correct JCL syntax. See *Data Sharing: Planning and Administration* for more information.

To accomplish the check, compare the expanded JCL in the SYSOUT output against the correct JCL provided in *OS/390 MVS JCL User's Guide* or *OS/390 MVS JCL Reference*. Then, take the member name of the erroneous JCL procedure also provided in the SYSOUT to the system programmer who maintains your procedure libraries. After finding out which proclib contains the JCL in question, locate the procedure and correct it.

### Starting after an abend

Starting DB2 after it abends is different from starting it after the command STOP DB2 has been issued. After STOP DB2, the system finishes its work in an orderly way and takes a shutdown checkpoint before stopping. When DB2 is restarted, it uses information from the system checkpoint and recovery log to determine the system status at shutdown.

When a power failure occurs, DB2 abends without being able to finish its work or take a shutdown checkpoint. When DB2 is restarted after an abend, it refreshes its knowledge of its status at termination using information on the recovery log and notifies the operator of the status of various units of recovery.

You can indicate that you want DB2 to postpone some of the backout work traditionally performed during system restart. You can delay the backout of long running units of recovery (URs) using installation options LIMIT BACKOUT and BACKOUT DURATION on panel DSNTIPN. For a description of these installation parameters, see Chapter 2 of *DB2 Installation Guide*.

Normally, the restart process resolves all inconsistent states. In some cases, you have to take specific steps to resolve inconsistencies. There are steps you can take to prepare for those actions. For example, you can limit the list of table spaces that are recovered automatically when DB2 is started. For an explanation of the causes of database inconsistencies and how you can prepare to recover from them, see “Chapter 19. Restarting DB2 after termination” on page 347.

## Stopping DB2

Before stopping, all DB2-related write to operator with reply (WTOR) messages must receive replies. Then one of the following commands terminates the subsystem:

```
-STOP DB2 MODE(QUIESCE)
-STOP DB2 MODE(FORCE)
```

For the effects of the QUIESCE and FORCE options, see “Normal termination” on page 347. In a data sharing environment, see *Data Sharing: Planning and Administration*.

The following messages are returned:

```
DSNY002I - SUBSYSTEM STOPPING
DSN9022I - DSNYASCP '-STOP DB2' NORMAL COMPLETION
DSN3104I - DSN3EC00 - TERMINATION COMPLETE
```

Before DB2 can be restarted, the following message must also be returned to the MVS console that is authorized to enter the START DB2 command:

```
DSN3100I - DSN3EC00 - SUBSYSTEM ssnm READY FOR -START COMMAND
```

If the STOP DB2 command is not issued from an MVS console, messages DSNY002I and DSN9022I are not sent to the IMS or CICS master terminal operator. They are routed only to the MVS console that issued the START DB2 command.

## Submitting work to be processed

An application program running under TSO, IMS, or CICS can make use of DB2 resources by executing embedded SQL statements. How to run application programs from those environments is explained under:

“Running TSO application programs” on page 259
“Running IMS application programs” on page 260
“Running CICS application programs” on page 261
“Running batch application programs” on page 261
“Running application programs using CAF” on page 262
“Running application programs using RRSAF” on page 263

In each case, there are some conditions that the application program must meet to embed SQL statements and to authorize the use of DB2 resources and data.

All application programming defaults, including the subsystem name that the programming attachments discussed here use, are in the DSNHDECP load module. Make sure your JCL specifies the proper set of program libraries.

## Using DB2I (DB2 Interactive)

Using the interactive program DB2I, you can run application programs and perform many DB2 operations by entering values on panels. DB2I runs under TSO using ISPF (Interactive System Productivity Facility) services. To use it, follow your local procedures for logging on to TSO, and enter ISPF. The DB2I menu is shown in Part 1 of *DB2 Application Programming and SQL Guide*.

You control each operation by entering the parameters that describe it on the panels provided. DB2 also provides help panels to:

- Explain how to use each operation
- Provide the syntax for and examples of DSN subcommands, DB2 operator commands, and DB2 utility control statements

To access the help panels, press the HELP PF key. (The key can be set locally, but typically is PF1.)

## Running TSO application programs

To run TSO application programs:

1. Log on.
2. Enter the DSN command.
3. Respond to the prompt by entering the RUN subcommand.

The following example runs application program DSN8BC3. The program is in library *prefix*.RUNLIB.LOAD, the name assigned to the load module library.

```
DSN SYSTEM (subsystem-name)
RUN PROGRAM (DSN8BC3) PLAN(DSN8BH71) LIB ('prefix.RUNLIB.LOAD')
END
```

A TSO application program that you run in a DSN session must be link-edited with the TSO language interface program (DSNELI). The program cannot include IMS DL/I calls because that requires the IMS language interface module (DFSLI000).

The terminal monitor program (TMP) attaches the DB2-supplied DSN command processor, which in turn attaches the application program.

The DSN command starts a DSN session, which in turn provides a variety of subcommands and other functions. The DSN subcommands are:

**ABEND**
Causes the DSN session to terminate with a DB2 X'04E' abend completion code and with a DB2 abend reason code of X'00C50101'

**BIND PACKAGE**
Generates an application package

**BIND PLAN**
Generates an application plan

**DCLGEN**
Produces SQL and host language declarations

**END** Ends the DB2 connection and returns to TSO

**FREE PACKAGE**
Deletes a specific version of a package

**FREE PLAN**
Deletes an application plan

**REBIND PACKAGE**
Regenerates an existing package

**REBIND PLAN**
Regenerates an existing plan

**RUN** Executes a user application program

**SPUFI** Invokes a DB2I facility for executing SQL statements not embedded in an application program

You can also issue the following DB2 and TSO commands from a DSN session:
- Any TSO command except TIME, TEST, FREE, and RUN.
- Any DB2 command except START DB2. For a list of those commands, see “DB2 operator commands” on page 250.

DB2 uses the following sources to find an authorization for access by the application program. DB2 checks the first source listed; if it is unavailable, it checks the second source, and so on.
1. RACF USER parameter supplied at logon
2. TSO logon user ID
3. Site-chosen default authorization ID
4. IBM-supplied default authorization ID

Either the RACF USER parameter or the TSO user ID can be modified by a locally defined authorization exit routine.

## Running IMS application programs

To run IMS application programs, enter transactions from IMS terminals.

Application programs that contain SQL statements run in message processing program (MPP), batch message processing (BMP), Fast Path regions, or IMS batch regions.

The program must be link-edited with the IMS language interface module (DFSLI000). It can write to and read from other database management systems using the distributed data facility, in addition to accessing DL/I and Fast Path resources.

DB2 checks whether the authorization ID provided by IMS is valid. For message-driven regions, IMS uses the SIGNON-ID or LTERM as the authorization ID. For non-message-driven regions and batch regions, IMS uses the ASXBUSER field (if RACF or another security package is active). The ASXBUSER field is defined by MVS as 7 characters. If the ASXBUSER field contains binary zeros or blanks (RACF or another security package is not active), IMS uses the PSB name instead. See “Chapter 12. Controlling access to a DB2 subsystem” on page 169 for more information about DB2 authorization IDs.

An IMS terminal operator probably notices few differences between application programs that access DB2 data and programs that access DL/I data because no messages relating to DB2 are sent to a terminal operator by IMS. However, your program can signal DB2 error conditions with a message of your choice. For example, at the program’s first SQL statement, it receives an SQL error code if the resources to run the program are not available or if the operator is not authorized to use the resources. The program can interpret the code and issue an appropriate message to the operator.

***Running IMS batch work:*** You can run batch DL/I jobs to access DB2 resources; DB2-DL/I batch support uses the IMS attach package.

See Part 5 of *DB2 Application Programming and SQL Guide* for more information about application programs and DL/I batch. See *IMS Application Programming: Design Guide* for more information about recovery and DL/I batch.

## Running CICS application programs

To run CICS applications, enter transactions from CICS terminals. You can also invoke CICS transactions by using the CICS transaction-invocation stored procedure. For information about this stored procedure, see “The CICS transaction invocation stored procedure (DSNACICS)” on page 1087.

CICS transactions that issue SQL statements must be link-edited with the CICS attachment facility language interface module, DSNCLI, and the CICS command language interface module. CICS application programs can issue SQL, DL/I, or CICS commands. After CICS has connected to DB2, any authorized CICS transaction can issue SQL requests that can write to and read from multiple DB2 instances using the distributed data facility. The application programs run as CICS applications.

DB2 checks an authorization ID related to the transaction against a plan assigned to it. The authorization ID for the transaction can be the operator ID, terminal ID, transaction ID, RACF-authenticated USERID, or another identifier explicitly provided by the resource control table (RCT). See “Chapter 12. Controlling access to a DB2 subsystem” on page 169 for more information about DB2 authorization IDs.

## Running batch application programs

Batch DB2 work can run in background under the TSO terminal monitor program (TMP) or in an IMS batch message processing (BMP) region. IMS batch regions can issue SQL statements.

Batch work is run in the TSO background under the TSO terminal monitor program (TMP). The input stream can invoke TSO command processors, particularly the DSN command processor for DB2, and can include DSN subcommands such as RUN. The following is an example of a TMP job:

```
//jobname  JOB  USER=SYSOPR ...
//GO       EXEC PGM=IKJEFT01,DYNAMNBR=20
.
user DD statements
.
//SYSTSPRT DD   SYSOUT=A
//SYSTSIN  DD   *
DSN SYSTEM (ssid)
.
subcommand  (for example, RUN)
.
END
/*
```

In the example,

- IKJEFT01 identifies an entry point for TSO TMP invocation. Alternate entry points defined by TSO are also available to provide additional return code and ABEND termination processing options. These options permit the user to select the actions to be taken by the TMP upon completion of command or program execution.

  Because invocation of the TSO TMP using the IKJEFT01 entry point might not be suitable for all user environments, refer to the TSO publications to determine which TMP entry point provides the termination processing options best suited to your batch execution environment.
- USER=SYSOPR identifies the user ID (SYSOPR in this case) for authorization checks.
- DYNAMNBR=20 indicates the maximum number of data sets (20 in this case) that can be dynamically allocated concurrently.
- MVS checkpoint and restart facilities do not support the execution of SQL statements in batch programs invoked by RUN. If batch programs stop because of errors, DB2 backs out any changes made since the last commit point. For information on backup and recovery, see “Chapter 21. Backing up and recovering databases” on page 373. For an explanation of backing out changes to data when a batch program run in the TSO background abends, see Part 5 of *DB2 Application Programming and SQL Guide*.
- (*ssid*) is the subsystem name or group attachment name.

## Running application programs using CAF

**General-use Programming Interface**

The call attachment facility (CAF) allows you to customize and control your execution environments more extensively than the TSO, CICS, or IMS attachment facilities. Programs executing in TSO foreground or TSO background can use either the DSN session or CAF; each has advantages and disadvantages. MVS batch and started task programs can use only CAF.

It is also possible for IMS batch applications to access DB2 databases through CAF, though this method does not coordinate the commitment of work between the IMS and DB2 systems. We highly recommend that you use the DB2 DL/I batch support for IMS batch applications.

In order to use CAF, you must first make available a load module known as the call attachment language interface or DSNALI. When the language interface is available, your program can use CAF in two ways:

- Implicitly, by including SQL statements or IFI calls in your program just as you would any program
- Explicitly, by writing CALL DSNALI statements

For an explanation of CAF's capabilities and how to use it, see Part 6 of *DB2 Application Programming and SQL Guide*.

**End of General-use Programming Interface**

## Running application programs using RRSAF

**General-use Programming Interface**

The Recoverable Resource Manager Services attachment facility (RRSAF) is a DB2 attachment facility that relies on an OS/390 component called OS/390 Transaction Management and Recoverable Resource Manager Services (OS/390 RRS). OS/390 RRS provides system-wide services for coordinating two-phase commit operations across MVS products.

Before you can run an RRSAF application, OS/390 RRS must be started. OS/390 RRS runs in its own address space and can be started and stopped independently of DB2.

Applications that use RRSAF must:

- Call DSNRLI to invoke RRSAF functions. Those functions establish a connection between DB2 and OS/390 RRS and allocate DB2 resources.
- Link-edit or load the RRSAF language interface module, DSNRLI.

See "Controlling OS/390 RRS connections" on page 304 for a description of how applications connect to DB2 using RRSAF. For an explanation of RRSAF's capabilities and how to use it, see Part 6 of *DB2 Application Programming and SQL Guide*.

**End of General-use Programming Interface**

## Receiving messages

DB2 message identifiers have the form DSN*cxxxt*, where:

**DSN** Is the unique DB2 message prefix.

*c* Is a 1-character code identifying the DB2 subcomponent that issued the message. For example:

| | |
|---|---|
| **M** | IMS attachment facility |
| **U** | Utilities |

*xxx* Is the message number

*t* Is the message type, with these values and meanings:

| | |
|---|---|
| **A** | Immediate action |
| **D** | Immediate decision |
| **E** | Eventual action |
| **I** | Information only |

See *DB2 Messages and Codes* for an expanded description of message types.

A command prefix, identifying the DB2 subsystem, follows the message identifier, except in messages from the CICS and IMS attachment facilities (subcomponents C for CICS Version 3 and below, 2 for CICS Version 4 and above, or M for IMS). CICS attachment facility messages identify the sending CICS subsystem and are sent to the MVS console, the CICS terminal, or the CICS transient data destination specified in the resource control table (RCT).

The IMS attachment facility issues messages that are identified as SSNM*xxxx* and as DFS*xxxx*. The DFS*xxxx* messages are produced by IMS, under which the IMS attachment facility operates.

## Receiving unsolicited DB2 messages

Unsolicited subsystem messages are sent to the MVS console issuing the START DB2 command, or to consoles assigned the routing codes that were listed in the DSNZP*xxx* module when installing DB2. But the following messages from the IMS and the CICS attachment facilities are exceptions to that rule:

- Specific IMS attachment facility messages are sent to the IMS master terminal.
- Unsolicited CICS messages are sent to the transient data entries specified in the RCT (ERRDEST).
- CICS statistics messages that are issued because of shutdown are sent to the transient data entry specified in the RCT (SHDDEST).

Some DB2 messages sent to the MVS console are defined as critical using the MVS/WTO descriptor code (11). This code signifies “critical eventual action requested” by DB2. Preceded by an at sign (@) or an asterisk (*), critical DB2 messages remain on the screen until specifically deleted. This prevents them from being missed by the operator, who is required to take a specific action.

## Determining operational control

Table 61 summarizes the operational control that is available at the operator console or terminal.

*Table 61. Operational control summary*

| Type of Operation | MVS Console | TSO Terminal | IMS Master Terminal | Authorized CICS Terminal |
|---|---|---|---|---|
| Issue DB2 commands and receive replies | Yes | Yes[1] | Yes[1] | Yes[1] |
| Receive DB2 unsolicited output | Yes | No | No | No |
| Issue IMS commands | Yes[2] | No | Yes | No |
| Receive IMS attachment facility unsolicited output | No[3] | No | Yes | No |
| Issue CICS commands | Yes[4] | No | No | Yes |
| Receive CICS attachment facility unsolicited output | No[3] | No | No | Yes[5] |

*Table 61. Operational control summary (continued)*

| Type of Operation | MVS Console | TSO Terminal | IMS Master Terminal | Authorized CICS Terminal |
|---|---|---|---|---|

**Notes:**

1. Except START DB2. Commands issued from IMS must have the prefix /SSR. Commands issued from CICS must have the prefix DSNC.
2. Using outstanding WTOR.
3. “Attachment facility unsolicited output” does not include “DB2 unsolicited output”; for the latter, see “Receiving unsolicited DB2 messages” on page 264.
4. Use the MVS command MODIFY *jobname*, *CICS command*. The MVS console must already be defined as a CICS terminal.
5. Specify the output destination for the unsolicited output of the CICS attachment facility in the RCT.

# Chapter 17. Monitoring and controlling DB2 and its connections

The information under this heading, up to “Controlling IMS connections” on page 295, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

“Chapter 16. Basic operation” on page 249 tells you how to start DB2, submit work to be processed, and stop DB2. The following operations, described in this chapter, require more understanding of what DB2 is doing:

“Controlling DB2 databases and buffer pools”
“Controlling user-defined functions” on page 277
“Controlling DB2 utilities” on page 278
“Controlling the IRLM” on page 280

This chapter also introduces the concept of a *thread*, a DB2 structure that makes the connection between another subsystem and DB2. A thread describes an application’s connection, traces its progress, and delimits its accessibility to DB2 resources and services. Most DB2 functions execute under a thread structure. The use of threads in making, monitoring, and breaking connections is described in the following sections:

“Monitoring threads” on page 283
“Controlling TSO connections” on page 284
“Controlling CICS connections” on page 287
“Controlling IMS connections” on page 295
“Controlling OS/390 RRS connections” on page 304
“Controlling connections to remote systems” on page 307

“Controlling traces” on page 326, tells you how to start and stop traces, and points to other books for help in analyzing their results.

“Controlling the resource limit facility (governor)” on page 328, tells how to start and stop the governor, and how to display its current status.

A final section, “Changing subsystem parameter values” on page 329, tells how to change subsystem parameters dynamically even when DB2 is running.

**Important:** Examples of commands in this chapter do not necessarily illustrate all the available options. For the complete syntax of any command or utility, see *DB2 Command Reference* or *DB2 Utility Guide and Reference*. For data sharing considerations, see *DB2 Data Sharing: Planning and Administration*.

## Controlling DB2 databases and buffer pools

DB2 databases are controlled by the following commands:

**START DATABASE**
Makes a database, or individual partitions, available. Also removes pages from the logical page list (LPL). For its use, see “Starting databases” on page 268.

**DISPLAY DATABASE**
Displays status, user, and locking information for a database. For its use, see “Monitoring databases” on page 269.

**STOP DATABASE**

Makes a database, or individual partitions, unavailable after existing users have quiesced. DB2 also closes and deallocates the data sets. For its use, see “Stopping databases” on page 274.

The START and STOP DATABASE commands can be used with the SPACENAM and PART options to control table spaces, index spaces, or partitions. For example, the following command starts two partitions of table space DSN8S71E in the database DSN8D71A:

```
-START DATABASE (DSN8D71A) SPACENAM (DSN8S71E) PART (1,2)
```

# Starting databases

The command START DATABASE (*) starts all databases for which you have the STARTDB privilege. The privilege can be explicitly granted, or can belong implicitly to a level of authority (DBMAINT and above, as shown in Figure 8 on page 109). The command starts the database, but not necessarily all the objects it contains. Any table spaces or index spaces in a restricted mode remain in a restricted mode and are not started.

START DATABASE (*) does not start the DB2 directory (DSNDB01), the DB2 catalog (DSNDB06), or the DB2 work file database (called DSNDB07, except in a data sharing environment). These databases have to be started explicitly using the SPACENAM option. Also, START DATABASE (*) does not start table spaces or index spaces that have been explicitly stopped by the STOP DATABASE command.

The PART keyword of the command START DATABASE can be used to start individual partitions of a table space. It can also be used to start individual partitions of a partitioning index or logical partitions of a nonpartitioning index. The started or stopped state of other partitions is unchanged.

## Starting an object with a specific status

A database, table space, or an index space can be started with a specific status that limits access to it.

| **Status** | **Provides this access** |
|---|---|
| **RW** | Read-write. This is the default value. |
| **RO** | Read only. You cannot change the data. |
| **UT** | Utility only. The object is available only to the DB2 utilities. |

Databases, table spaces, and index spaces are started with RW status when they are created. You can make any of them unavailable by using the command STOP DATABASE. DB2 can also make them unavailable when it detects an error.

In cases when the object was explicitly stopped, you can make them available again using the command START DATABASE. For example, the following command starts all table spaces and index spaces in database DSN8D71A for read-only access:

```
-START DATABASE (DSN8D71A) SPACENAM(*) ACCESS(RO)
```

The system responds with this message:

```
DSN9022I - DSNTDDIS '-START DATABASE' NORMAL COMPLETION
```

## Starting a table space or index space that has restrictions

DB2 can make an object unavailable for a variety of reasons. Typically, in those cases, the data is unreliable and the object needs some attention before it can be started. An example of such a restriction is when the table space is placed in *copy pending* status. That status makes a table space or partition unavailable until an image copy has been made of it.

These restrictions are a necessary part of protecting the integrity of the data. **If you start an object that has restrictions, the data in the object might not be reliable.**

However, in certain circumstances, it might be reasonable to force availability. For example, a table might contain test data whose consistency is not critical. In those cases, the objects can be started by using the ACCESS(FORCE) option of START DATABASE. For example:

```
-START DATABASE (DSN8D71A) SPACENAM (DSN8S71E) ACCESS(FORCE)
```

The command releases most restrictions for the named objects. These objects must be explicitly named in a list following the SPACENAM option.

DB2 cannot process the START DATABASE ACCESS(FORCE) request if postponed abort or indoubt URs exist. The *restart pending* (RESTP) status and the *advisory restart pending* (AREST) status remain in effect until either automatic backout processing completes or until you perform one of the following actions:

- Issue the RECOVER POSTPONED command to complete backout activity.
- Issue the RECOVER POSTPONED CANCEL command to cancel all of the postponed abort units of recovery.
- Conditionally restart or cold start DB2.

For more information on resolving postponed units of recovery, see “Resolving postponed units of recovery” on page 355.

## Monitoring databases

You can use the command DISPLAY DATABASE to obtain information about the status of databases and the table spaces and index spaces within each database. If applicable, the output also includes information about physical I/O errors for those objects. Use DISPLAY DATABASE as follows:

```
-DISPLAY DATABASE (dbname)
```

This results in the following messages:

```
11:44:32 DSNT360I - ****************************************************
11:44:32 DSNT361I - *  DISPLAY DATABASE SUMMARY
11:44:32            *      report_type_list
11:44:32 DSNT360I - ****************************************************
11:44:32 DSNT362I -     DATABASE = dbname   STATUS = xx
                           DBD LENGTH = yyyy

11:44:32 DSNT397I -

NAME     TYPE PART STATUS           PHYERRLO  PHYERRHI CATALOG  PIECE
-------- ---- ---- ---------------- --------- -------- -------- -----
D1       TS        RW,UTRO
D2       TS        RW
D3       TS        STOP
D4       IX        RO
D5       IX        STOP
D6       IX        UT
LOB1     LS        RW
******* DISPLAY OF DATABASE dbname ENDED **********************
11:45:15 DSN9022I - DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

In the preceding messages:

- *Report_type_list* indicates which options were included when the DISPLAY DATABASE command was issued. See Chapter 2 of *DB2 Command Reference* for detailed descriptions of options.
- *dbname* is an 8-byte character string indicating the database name. The pattern-matching character, *, is allowed at the beginning, middle, and end of *dbname*.
- STATUS is a combination of one or more status codes delimited by a comma. The maximum length of the string is 18 characters. If the status exceeds 18 characters, those characters are wrapped onto the next status line. Anything that exceeds 18 characters on the second status line is truncated. See Chapter 2 of *DB2 Command Reference* for a list of status codes and their descriptions.

You can use the pattern-matching character, *, in the commands DISPLAY DATABASE, START DATABASE, and STOP DATABASE. The pattern-matching character can be used in the beginning, middle, and end of the database and table space names. The keyword ONLY can be added to the command DISPLAY DATABASE. When ONLY is specified with the DATABASE keyword but not the SPACENAM keyword, all other keywords except RESTRICT, LIMIT, and AFTER are ignored. Use DISPLAY DATABASE as follows:

```
-DISPLAY DATABASE (*S*DB*) ONLY
```

This results in the following messages:

```
11:44:32 DSNT360I - ****************************************************
11:44:32 DSNT361I - *  DISPLAY DATABASE SUMMARY
11:44:32            *      GLOBAL
11:44:32 DSNT360I - ****************************************************
11:44:32 DSNT362I -     DATABASE = DSNDB01   STATUS = RW
                           DBD LENGTH = 8066
11:44:32 DSNT360I - ****************************************************
11:44:32 DSNT362I -     DATABASE = DSNDB04  STATUS = RW
                           DBD LENGTH = 21294
11:44:32 DSNT360I - ****************************************************
11:44:32 DSNT362I -     DATABASE = DSNDB06  STATUS = RW
                           DBD LENGTH = 32985
11:45:15 DSN9022I - DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

In the preceding messages:

- DATABASE (*S*DB*) displays databases that begin with any letter, have the letter S followed by any letters, then the letters DB followed by any letters.
- ONLY restricts the display to databases names that fit the criteria.

See Chapter 2 of *DB2 Command Reference* for detailed descriptions of these and other options on the DISPLAY DATABASE command.

You can use the RESTRICT(REFP) option of the DISPLAY DATABASE command to limit the display to a table space or partition in refresh pending (REFP) status. For information about resetting a restrictive status, see Appendix C of *DB2 Utility Guide and Reference*.

You can use the ADVISORY option on the DISPLAY DATABASE command to limit the display to table spaces or indexes that require some corrective action. Use the DISPLAY DATABASE ADVISORY command without the RESTRICT option to determine when:

- An index space is in the informational copy pending (ICOPY) advisory status
- A base table space is in the auxiliary warning (AUXW) advisory status

For information about resetting an advisory status, see Appendix C of *DB2 Utility Guide and Reference*.

## Obtaining information about application programs

You can obtain various kinds of information about application programs using particular databases or table or index spaces with the DISPLAY DATABASE command. This section describes how you can identify who or what is using the object and what locks are being held on the objects.

***Who and what is using the object?*** You can obtain the following information:

- The names of the application programs currently using the database or space
- The authorization IDs of the users of these application programs
- The logical unit of work IDs of the database access threads accessing data on behalf of the remote locations specified.

To obtain this information, issue a command, for example, that names partitions 2, 3, and 4 in table space TPAUGF01 in database DBAUGF01:

```
-DISPLAY DATABASE (DBAUGF01) SPACENAM (TPAUGF01) PART (2,3,4) USE
```

DB2 returns a list similar to this one:

```
DSNT360I : ***********************************
DSNT361I : *  DISPLAY DATABASE SUMMARY
           *    GLOBAL USE
DSNT360I : ***********************************
DSNT362I :     DATABASE = DBAUGF01  STATUS = RW
               DBD LENGTH = 8066
DSNT397I :
NAME     TYPE PART STATUS             CONNID   CORRID       USERID
-------- ---- ---- ------------------ -------- ------------ --------
TPAUGF01 TS    002 RW                 BATCH    S3341209     ADMF001
    -                  MEMBER NAME V61A
TPAUGF01 TS    003 RW                 BATCH    S3341209     ADMF001
    -                  MEMBER NAME V61A
TPAUGF01 TS    004 RW                 BATCH    S3341209     ADMF001
    -                  MEMBER NAME V61A
******* DISPLAY OF DATABASE DBAUGF01 ENDED      **********************
DSN9022I : DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

***Which programs are holding locks on the objects?*** To determine which application programs are currently holding locks on the database or space, issue a command like the following, which names table space TSPART in database DB01:

```
-DISPLAY DATABASE(DB01) SPACENAM(TSPART) LOCKS
```

DB2 returns a list similar to this one:

```
17:45:42 DSNT360I - ****************************************************
17:45:42 DSNT361I - *  DISPLAY DATABASE SUMMARY
17:45:42            *    GLOBAL LOCKS
17:45:42 DSNT360I - ****************************************************
17:45:42 DSNT362I -     DATABASE = DB01  STATUS = RW
17:45:42                DBD LENGTH = yyyy
17:45:42 DSNT397I -
 NAME     TYPE PART STATUS             CONNID   CORRID       LOCKINFO
 -------- ---- ---- ------------------ -------- ------------ ---------
 TSPART   TS    01  RW                 LSS001   DSN2SQL      H-IX,P,C
 TSPART   TS    02  RW                 LSS001   DSN2SQL      H-IX,P,C
 TSPART   TS    03  RW                 LSS001   DSN2SQL      H-IX,P,C
 TSPART   TS    04  RW                 LSS001   DSN2SQL      H-IX,P,C
 ******* DISPLAY OF DATABASE DB01     ENDED    **********************
 17:45:44 DSN9022I . DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

For an explanation of the field LOCKINFO, see message DSNT396I in Part 2 of *DB2 Messages and Codes*.

Use the LOCKS ONLY keywords on the DISPLAY DATABASE command to display only spaces that have locks. The LOCKS keyword can be substituted with USE, CLAIMERS, LPL, or WEPR to display only databases that fit the criteria. Use DISPLAY DATABASE as follows:

```
-DISPLAY DATABASE (DSNDB06) SPACENAM(*) LOCKS ONLY
```

This results in the following messages:

```
11:44:32 DSNT360I - ****************************************************
11:44:32 DSNT361I - *  DISPLAY DATABASE SUMMARY
11:44:32            *      GLOBAL LOCKS
11:44:32 DSNT360I - ****************************************************
11:44:32 DSNT362I -     DATABASE = DSNDB06   STATUS = RW
                           DBD LENGTH = 60560
11:44:32 DSNT397I -
NAME     TYPE PART STATUS             CONNID   CORRID       LOCKINFO
-------- ---- ---- ------------------ -------- ------------ ---------
SYSDBASE TS        RW                 DSN     020.DBCMD 06 H-IS,S,C
******* DISPLAY OF DATABASE DSNDB06  ENDED      **********************
11:45:15 DSN9022I - DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

See Chapter 2 of *DB2 Command Reference* for detailed descriptions of these and other options of the DISPLAY DATABASE command.

## Obtaining information about pages in error

There are two ways that pages can be in error:

- A page is logically in error if its problem can be fixed without redefining new disk tracks or volumes. For example, if DB2 cannot write a page to disk because of a connectivity problem, the page is logically in error. DB2 inserts entries for pages that are logically in error in a *logical page list* (LPL).
- A page is physically in error if there are physical errors, such as device errors. Such errors appear on the *write error page range* (WEPR). The range has a low and high page, which are the same if only one page has errors.

If the cause of the problem is undetermined, the error is first recorded in the LPL. If recovery from the LPL is unsuccessful, the error is then recorded on the error page range.

Write errors for large object data type (LOB) table spaces defined with LOG NO cause the unit of work to be rolled back. Because the pages are written during normal deferred write processing, they can appear in the LPL and WEPR. The LOB data pages for a LOB table space with the LOG NO attribute are not written to LPL or WEPR. The space map pages are written during normal deferred write processing and can appear in the LPL and WEPR.

A program that tries to read data from a page listed on the LPL or WEPR receives an SQLCODE for “resource unavailable”. To access the page (or pages in the error range), you must first recover the data from the existing database copy and the log.

***Displaying the logical page list:*** You can check the existence of LPL entries by issuing the DISPLAY DATABASE command with the LPL option. The ONLY option restricts the output to objects that have LPL pages. For example:

```
-DISPLAY DATABASE(DBFW8401) SPACENAM(*) LPL ONLY
```

Output similar to the following is produced:

```
DSNT360I = ***********************************************************
DSNT361I = *  DISPLAY DATABASE SUMMARY
           *    GLOBAL LPL
DSNT360I = ***********************************************************
DSNT362I =     DATABASE = DBFW8401  STATUS = RW,LPL
               DBD LENGTH = 8066
DSNT397I =
NAME     TYPE PART STATUS             LPL PAGES
-------- ---- ---- ------------------ ------------------
TPFW8401 TS    01  RW,LPL             000000-000004
ICFW8401 IX    01  RW,LPL             000000,000003
IXFW8402 IX        RW,LPL             000000,000003-000005
----                                  000007,000008-00000B
----                                  000080-000090
******* DISPLAY OF DATABASE DBFW8401 ENDED      **********************
DSN9022I = DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

The display indicates that the pages listed in the LPL PAGES column are unavailable for access. For the syntax and description of DISPLAY DATABASE, see Chapter 2 of *DB2 Command Reference*.

***Removing pages from the LPL:*** When an object has pages on the LPL, there are several ways to remove those pages and make them available for access when DB2 is running:

- Start the object with access (RW) or (RO). That command is valid even if the table space is already started.

  When you issue the command START DATABASE, you see message DSNI006I, indicating that LPL recovery has begun. Message DSNI022I is issued periodically to give you the progress of the recovery. When recovery is complete, you see DSNI021I.

  | When you issue the command START DATABASE for a LOB table space that is defined as LOG NO, and DB2 detects log records required for LPL recovery are missing due to the LOG NO attribute, the LOB table space is placed in AUXW status and the LOB is invalidated.
- Run the RECOVER or REBUILD INDEX utility on the object.

The only exception to this is when a logical partition of a nonpartitioned index has both LPL and RECP status. If you want to recover the logical partition using REBUILD INDEX with the PART keyword, you must first use the command START DATABASE to clear the LPL pages.

- Run the LOAD utility with the REPLACE option on the object.
- Issue an SQL DROP statement for the object.

Only the following utilities can be run on an object with pages in the LPL:

LOAD with the REPLACE option
MERGECOPY
REBUILD INDEX
RECOVER, *except*:
  RECOVER...PAGE
  RECOVER...ERROR RANGE
REPAIR with the SET statement
REPORT

***Displaying a write error page range:*** Use DISPLAY DATABASE to display the range of error pages. For example, this command:

```
-DISPLAY DATABASE (DBPARTS) SPACENAM (TSPART01) WEPR
```

might display a list such as this:

```
11:44:32 DSNT360I - ****************************************************
11:44:32 DSNT361I - *  DISPLAY DATABASE SUMMARY
11:44:32            *          GLOBAL WEPR
11:44:32 DSNT360I - ****************************************************
11:44:32 DSNT362I -     DATABASE = DBPARTS  STATUS = RW
                        DBD LENGTH = yyyy
11:44:32 DSNT397I -

NAME     TYPE PART STATUS           PHYERRLO  PHYERRHI CATALOG  PIECE
-------- ---- ---- ---------------- --------  -------- -------- -----
TSPART01 TS   001  RW,UTRO          00000002  00000004 DSNCAT   000
TSPART01 TS   002  RW,UTRO          00000009  00000013 DSNCAT   001
TSPART01 TS   003  RO
TSPART01 TS   004  STOP
TSPART01 TS   005  UT
******* DISPLAY OF DATABASE DBPARTS ENDED       **********************
11:45:15 DSN9022I - DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

In the previous messages:

- PHYERRLO and PHYERRHI identify the range of pages that were being read when the I/O errors occurred. PHYERRLO is an 8-digit hexadecimal number representing the lowest page found in error, while PHYERRHI represents the highest page found in error.
- PIECE, a 3-digit integer, is a unique identifier for the data set supporting the page set that contains physical I/O errors.

For additional information about this list, see the description of message DSNT392I in Part 2 of *DB2 Messages and Codes*.

## Stopping databases

Databases, table spaces, and index spaces can be made unavailable with the STOP DATABASE command. You can also use STOP DATABASE with the PART option to stop the following types of partitions:

- Physical partitions within a table space

- Physical partitions within an index space
- Logical partitions within a nonpartitioning index associated with a partitioned table space.

This prevents access to individual partitions within a table or index space while allowing access to the others. When you specify the PART option with STOP DATABASE on physically partitioned spaces, the data sets supporting the given physical partitions are closed and do not affect the remaining partitions. However, STOP DATABASE with the PART option does not close data sets associated with logically partitioned spaces. To close these data sets, you must execute STOP DATABASE without the PART option.

The AT(COMMIT) option of STOP DATABASE stops objects quickly. The AT(COMMIT) option interrupts threads that are bound with RELEASE(DEALLOCATE) and is useful when thread reuse is high.

If you specify AT(COMMIT), DB2 takes over access to an object when all jobs release their claims on it and when all utilities release their drain locks on it. If you do not specify AT(COMMIT), the objects are not stopped until all existing applications have deallocated. New transactions continue to be scheduled, but they receive SQLCODE -904 SQLSTATE '57011' (resource unavailable) on the first SQL statement that references the object or when the plan is prepared for execution. STOP DATABASE waits for a lock on an object that it is attempting to stop. If the wait time limit for locks (15 timeouts) is exceeded, then the STOP DATABASE command terminates abnormally and leaves the object in stop pending status (STOPP).

Database DSNDB01 and table spaces DSNDB01.DBD01 and DSNDB01.SYSLGRNX must be started before stopping user-defined databases or the work file database. A DSNI003I message tells you that the command was unable to stop an object. You must resolve the problem indicated by this message and run the job again. If an object is in STOPP status, you must first issue the START DATABASE command to remove the STOPP status and then issue the STOP DATABASE command.

DB2 subsystem databases (catalog, directory, work file) can also be stopped. After the directory is stopped, installation SYSADM authority is required to restart it.

The following examples illustrate ways to use the command:

**-STOP DATABASE (*)**
Stops all databases for which you have STOPDB authorization, except the DB2 directory (DSNDB01), the DB2 catalog (DSNDB06), or the DB2 work file database (called DSNDB07, except in a data sharing environment), all of which must be stopped explicitly.

**-STOP DATABASE (***dbname***)**
Stops a database, and closes all of the data sets of the table spaces and index spaces in the database.

**-STOP DATABASE (***dbname, ...***)**
Stops the named databases and closes all of the table spaces and index spaces in the databases. If DSNDB01 is named in the database list, it should be last on the list because stopping the other databases requires that DSNDB01 be available.

**-STOP DATABASE (*dbname*) SPACENAM (*)**
: Stops and closes all of the data sets of the table spaces and index spaces in the database. The status of the named database does not change.

**-STOP DATABASE (*dbname*) SPACENAM (*space-name, ...*)**
: Stops and closes the data sets of the named table space or index space. The status of the named database does not change.

**-STOP DATABASE (*dbname*) SPACENAM (*space-name, ...*) PART(*integer*)**
: Stops and closes the specified partition of the named table space or index space. The status of the named database does not change. If the named index space is nonpartitioned, DB2 cannot close the specified logical partition.

The data sets containing a table space are closed and deallocated by the commands listed above.

## Altering buffer pools

DB2 maintains the buffer pool attributes that were defined during installation, such as buffer pool and hiperpool sizes, in the DB2 bootstrap data set. These attributes are the same each time DB2 starts.

You can use the ALTER BUFFERPOOL command to alter buffer pool attributes, including the buffer pool sizes, sequential steal thresholds, deferred write thresholds, parallel sequential thresholds, and hiperpool CASTOUT attributes for active or inactive buffer pools. Altered buffer pool values are stored and used until altered again.

See Chapter 2 of *DB2 Command Reference* for descriptions of the options you can use with this command. See “Tuning database buffer pools” on page 549 for guidance on using buffer pools and examples of ALTER BUFFERPOOL.

## Monitoring buffer pools

Use the DISPLAY BUFFERPOOL command to display the current status for one or more active or inactive buffer pools. You can request a summary or detail report.

For example, the following command:

```
-DISPLAY BUFFERPOOL(BP0)
```

might produce a summary report such as this:

```
-DIS BUFFERPOOL(BP0)
DSNB401I ! BUFFERPOOL NAME BP0, BUFFERPOOL ID 0, USE COUNT 27
DSNB402I ! VIRTUAL BUFFERPOOL SIZE = 100 BUFFERS
             ALLOCATED       =      500   TO BE DELETED   =        0
             IN-USE/UPDATED  =        0
DSNB406I ! CURRENT VIRTUAL BUFFERPOOL TYPE   = PRIMARY
           PENDING VIRTUAL BUFFERPOOL TYPE   = DATASPACE
           PAGE STEALING METHOD              = LRU
DSNB403I ! HIPERPOOL SIZE = 1000 BUFFERS, CASTOUT = YES
             ALLOCATED       =        0   TO BE DELETED   =        0
             BACKED BY ES    =        0
DSNB404I ! THRESHOLDS -
            VP SEQUENTIAL       = 80   HP SEQUENTIAL          = 75
            DEFERRED WRITE      = 85   VERTICAL DEFERRED WRT  = 10,15
            PARALLEL SEQUENTIAL = 50   ASSISTING PARALLEL SEQT=  0
DSN9022I ! DSNB1CMD '-DIS BUFFERPOOL' NORMAL COMPLETION
```

See Chapter 2 of *DB2 Command Reference* for descriptions of the options you can use with this command and the information you find in the summary and detail reports.

## Controlling user-defined functions

User-defined functions are extensions to the SQL language. You can invoke user-defined functions in an SQL statement wherever you can use expressions or built-in functions. User-defined functions, like stored procedures, run in WLM-established address spaces. For information on creating, defining and invoking user-defined functions, see *DB2 SQL Reference*. For information about working with WLM-established address spaces, see “Monitoring and controlling stored procedures” on page 320.

DB2 user-defined functions are controlled by the following commands described in Chapter 2 of *DB2 Command Reference*:

**START FUNCTION SPECIFIC**
Activates an external function that is stopped.

**DISPLAY FUNCTION SPECIFIC**
Displays statistics about external user-defined functions accessed by DB2 applications.

**STOP FUNCTION SPECIFIC**
Prevents DB2 from accepting SQL statements with invocations of the specified functions.

## Starting user-defined functions

The DB2 command START FUNCTION SPECIFIC activates an external function that is stopped. Built-in functions or user-defined functions that are sourced on another function cannot be started.

Use the START FUNCTION SPECIFIC command to activate all or a specific set of stopped external functions. For example, issue a command like the following, which starts functions USERFN1 and USERFN2 in the PAYROLL schema:

```
START FUNCTION SPECIFIC(PAYROLL.USERFN1,PAYROLL.USERFN2)
```

Output similar to the following is produced:

```
DSNX973I START FUNCTION SPECIFIC SUCCESSFUL FOR PAYROLL.USERFN1
DSNX973I START FUNCTION SPECIFIC SUCCESSFUL FOR PAYROLL.USERFN2
```

## Monitoring user-defined functions

The DB2 command DISPLAY FUNCTION SPECIFIC displays statistics about external user-defined functions accessed by DB2 applications. This command displays one output line for each function that has been accessed by a DB2 application. Information returned by this command reflect a dynamic status. By the time DB2 displays the information, it is possible that the status might have changed. Built-in functions or user-defined functions that are sourced on another function are not applicable to this command.

Use the DISPLAY FUNCTION SPECIFIC command to list the range of functions that are stopped because of a STOP FUNCTION SPECIFIC command. For example, issue a command like the following, which displays information about functions in the PAYROLL schema and the HRPROD schema:

```
DISPLAY FUNCTION SPECIFIC(PAYROLL.*,HRPROD.*)
```

Output similar to the following is produced:

```
DSNX975I DSNX9DIS - DISPLAY FUNCTION SPECIFIC REPORT FOLLOWS-

------ SCHEMA=PAYROLL
FUNCTION          STATUS   ACTIVE  QUEUED  MAXQUE  TIMEOUT  WLM_ENV
PAYRFNC1          STARTED       0       0       1        0  PAYROLL
PAYRFNC2          STOPQUE       0       5       5        3  PAYROLL
PAYRFNC3          STARTED       2       0       6        0  PAYROLL
USERFNC4          STOPREJ       0       0       1        0  SANDBOX

------ SCHEMA=HRPROD
FUNCTION          STATUS   ACTIVE  QUEUED  MAXQUE  TIMEOUT  WLM_ENV
HRFNC1            STARTED       0       0       1        0  HRFUNCS
HRFNC2            STOPREJ       0       0       1        0  HRFUNCS

DSNX9DIS DISPLAY FUNCTION SPECIFIC REPORT COMPLETE
```

## Stopping user-defined functions

The DB2 command STOP FUNCTION SPECIFIC prevents DB2 from accepting SQL statements with invocations of the specified functions. This command does not prevent SQL statements with invocations of the functions from running if they have already been queued or scheduled by DB2. You cannot stop built-in functions or user-defined functions that are sourced on another function. While the STOP FUNCTION SPECIFIC command is in effect, attempts to execute the stopped functions are queued.

Use the STOP FUNCTION SPECIFIC command to stop access to all or a specific set of external functions. For example, issue a command like the following, which stops functions USERFN1 and USERFN3 in the PAYROLL schema:

```
STOP FUNCTION SPECIFIC(PAYROLL.USERFN1,PAYROLL.USERFN3)
```

Output similar to the following is produced:

```
DSNX974I STOP FUNCTION SPECIFIC SUCCESSFUL FOR PAYROLL.USERFN1
DSNX974I STOP FUNCTION SPECIFIC SUCCESSFUL FOR PAYROLL.USERFN3
```

# Controlling DB2 utilities

You can run DB2 utilities against databases, table spaces, indexes, index spaces, and partitions.

DB2 utilities are classified into two groups: online and stand-alone. The online utilities require DB2 to be running and can be invoked in several different ways. The stand-alone utilities do not require DB2 to be up, and they can be invoked only by means of MVS JCL. The online utilities are described in Part 2 of *DB2 Utility Guide and Reference*, and the stand-alone utilities are described in Part 3 of *DB2 Utility Guide and Reference*.

## Starting online utilities

To start a DB2 utility, prepare an appropriate set of JCL statements for a utility job
| and include DB2 utility control statements in the input stream for that job. DB2
| utilities can dynamically create object lists from a pattern-matching expression and
| can dynamically allocate the data sets required to process those objects.

## Monitoring online utilities

The following commands for monitoring and changing DB2 utility jobs are described in Chapter 2 of *DB2 Command Reference*.

**ALTER UTILITY**
Alters parameter values of an active REORG utility.

**DISPLAY UTILITY**
Displays the status of utility jobs.

**TERM UTILITY**
Terminates a utility job before its normal completion.

**If a utility is not running**, you need to determine whether the type of utility access is allowed on an object of a specific status. Table 62 shows the compatibility of utility types and object status.

*Table 62. Compatibility of utility types and object status*

| Utility types ... | Can access objects started as ... |
|---|---|
| Read-only | RO |
| All | RW[1] |
| DB2 | UT |

Note 1: RW is the default access type for an object.

**To change the status of an object**, use the ACCESS option of the START DATABASE command to start the object with a new status. For example:

```
-START DATABASE (DSN8D61A) ACCESS(RO)
```

For more information on concurrency and compatibility of individual online utilities, see Part 2 of *DB2 Utility Guide and Reference*. For a general discussion controlling concurrency for utilities, see Part 5 (Volume 2) of *DB2 Administration Guide*.

## Stand-alone utilities

The following stand-alone utilities can be run only by means of MVS JCL:

DSN1CHKR
DSN1COPY
DSN1COMP
DSN1PRNT
DSN1SDMP
DSN1LOGP
DSNJLOGF
DSNJU003 (change log inventory)
DSNJU004 (print log map)

Most of the stand-alone utilities can be used while DB2 is running. However, for consistency of output, the table spaces and index spaces must be stopped first because these utilities do not have access to the DB2 buffer pools. In some cases, DB2 *must* be running or stopped before you invoke the utility. See Part 3 of *DB2 Utility Guide and Reference* for detailed environmental information about these utilities.

| Stand-alone utility job streams require that you code specific data set names in the
| JCL. To determine the fifth qualifier in the data set name, you need to query the
| DB2 catalog tables SYSIBM.SYSTABLEPART and SYSIBM.SYSINDEXPART to
| determine the IPREFIX column that corresponds to the required data set.

The *change log inventory utility* (DSNJU003) enables you to change the contents of the bootstrap data set (BSDS). This utility cannot be run while DB2 is running because inconsistencies could result. Use STOP DB2 MODE(QUIESCE) to stop the DB2 subsystem, run the utility, and then restart DB2 with the START DB2 command.

The *print log map utility* (DSNJU004) enables you to print the the bootstrap data set contents. The utility can be run when DB2 is active or inactive; however, when it is run with DB2 active, the user's JCL and the DB2 started task must both specify DISP=SHR for the BSDS data sets.

## Controlling the IRLM

The internal resource lock manager (IRLM) subsystem manages DB2 locks. The particular IRLM to which DB2 is connected is specified in DB2's load module for subsystem parameters. It is also identified as an MVS subsystem in the SYS1.PARMLIB member IEFSSN*xx*. That name is used as the IRLM procedure name (*irlmproc*) in MVS commands.

IMS and DB2 must use separate instances of IRLM.

***Data sharing:*** In a data sharing environment, the IRLM handles global locking, and each DB2 member has its own corresponding IRLM. See *DB2 Data Sharing: Planning and Administration* for more information about configuring IRLM in a data sharing environment.

You can use the following MVS commands to control the IRLM. *irlmproc* is the IRLM procedure name, and *irlmnm* is the IRLM subsystem name. See Chapter 2 of *DB2 Command Reference* for more information about these commands.

**MODIFY** *irlmproc***,ABEND,DUMP**
Abends the IRLM and generates a dump.

**MODIFY** *irlmproc***,ABEND,NODUMP**
Abends the IRLM but does not generate a dump.

**MODIFY** *irlmproc***,DIAG**
Initiates diagnostic dumps for IRLM subsystems in a data sharing group when there is a delay.

**MODIFY** *irlmproc***,SET**
Sets the maximum amount of CSA storage or the number of trace buffers used for this IRLM.

**MODIFY** *irlmproc***,STATUS**
Displays the status for the subsystems on this IRLM.

**START** *irlmproc*
Starts the IRLM.

**STOP** *irlmproc*
Stops the IRLM normally.

**TRACE CT,OFF,COMP=***irlmnm*
Stops IRLM tracing.

**TRACE CT,ON,COMP=***irlmnm*
Starts IRLM tracing for all subtypes (DBM,SLM,XIT,XCF).

**TRACE CT,ON,COMP=***irlmnm***,SUB=(***subname***)**
Starts IRLM tracing for a single subtype.

## Starting the IRLM

The IRLM must be available when DB2 starts or when DB2 abends with reason code X'00E30079'.

When DB2 is installed, you normally specify that the IRLM be started automatically. Then, if the IRLM is not available when DB2 is started, DB2 starts it, and periodically checks whether it is up before attempting to connect. If the attempt to start the IRLM fails, DB2 terminates.

If an automatic IRLM start has not been specified, start the IRLM before starting DB2, using the MVS START *irlmproc* command.

When started, the IRLM issues this message to the MVS console:

```
DXR117I irlmnm INITIALIZATION COMPLETE
```

Consider starting the IRLM manually if you are having problems starting DB2 for either of these reasons:

- An IDENTIFY or CONNECT to a data sharing group fails.
- DB2 experiences a failure that involves the IRLM.

When you start the IRLM manually, you can generate a dump to collect diagnostic information because IRLM does not stop automatically.

## Modifying the IRLM

You can use the following MVS commands to modify the IRLM. See Chapter 2 of *DB2 Command Reference* for more information about these commands.

**MODIFY** *irlmproc***,SET,CSA=***nnn*

Sets the maximum amount of CSA storage that this IRLM can use for lock control structures.

**MODIFY** *irlmproc***,SET,DEADLOCK=***nnnn*

Sets the time for the local deadlock detection cycle.

**MODIFY** *irlmproc***,SET,LTE=***nnnn*

Sets the number of LOCK HASH entries that this IRLM can use on the next connect to the XCF LOCK structure. Use only for data sharing.

**MODIFY** *irlmproc***,SET,TIMEOUT=***nnnn***,***subsystem-name*

Sets the timeout value for the specified DB2 subsystem. Display the *subsystem-name* by using MODIFY *irlmproc*,STATUS.

**MODIFY** *irlmproc***,SET,TRACE=***nnn*

Sets the maximum number of trace buffers used for this IRLM.

## Monitoring the IRLM connection

You can use the following MVS commands to monitor the IRLM connection. See Chapter 2 of *DB2 Command Reference* for more information about these commands.

**MODIFY** *irlmproc***,STATUS,***irlmnm*

Displays the status of a specific IRLM.

**MODIFY** *irlmproc***,STATUS,ALLD**

Displays the status of all subsystems known to this IRLM in the data sharing group.

**MODIFY** ***irlmproc*****,STATUS,ALLI**
Displays the status of all IRLMs known to this IRLM in the data sharing group.

**MODIFY** ***irlmproc*****,STATUS,MAINT**
Displays the maintenance levels of IRLM load module CSECTs for the specified IRLM instance.

**MODIFY** ***irlmproc*****,STATUS,STOR**
Displays the current and *high water* allocation for CSA and ECSA storage.

**MODIFY** ***irlmproc*****,STATUS,TRACE**
Displays information about trace types of IRLM subcomponents.

## Stopping the IRLM

If the IRLM is started automatically by DB2, it stops automatically when DB2 is stopped. If the IRLM is not started automatically, you must stop it after DB2 stops.

If you try to stop the IRLM while DB2 or IMS is still using it, you get the following message:

```
DXR105E irlmnm STOP COMMAND REJECTED. AN IDENTIFIED SUBSYSTEM
IS STILL ACTIVE
```

If that happens, issue the STOP *irlmproc* command again, when the subsystems are finished with the IRLM.

Or, if you must stop the IRLM immediately, enter the following command to force the stop:

```
MODIFY irlmproc,ABEND,NODUMP
```

The system responds with this message:

```
DXR165I KRLM TERMINATED VIA IRLM MODIFY COMMAND.
DXR121I KRLM END-OF-TASK CLEANUP SUCCESSFUL - HI-CSA 335K
-  HI-ACCT-CSA      0K
```

DB2 abends. An IMS subsystem using the IRLM does not abend and can be reconnected.

IRLM uses the MVS Automatic Restart Manager (ARM) services. However, it de-registers from ARM for normal shutdowns. IRLM registers with ARM during initialization and provides ARM with an event exit. The event exit must be in the link list. It is part of the IRLM DXRRL183 load module. The event exit will make sure that the IRLM name is defined to MVS when ARM restarts IRLM on a target MVS that is different from the failing MVS. The IRLM element name used for the ARM registration depends on the IRLM mode. For local mode IRLM, the element name is a concatenation of the IRLM subsystem name and the IRLM ID. For global mode IRLM, the element name is a concatenation of the IRLM data sharing group name, IRLM subsystem name, and the IRLM ID.

IRLM de-registers from ARM when one of the following events occurs:

- PURGE *irlmproc* is issued.
- MODIFY *irlmproc*,ABEND,NODUMP is issued.
- DB2 automatically stops IRLM.

The command MODIFY *irlmproc*,ABEND,NODUMP specifies that IRLM de-register from ARM before terminating, which prevents ARM from restarting IRLM. However, it does not prevent ARM from restarting DB2, and, if you set the automatic restart manager to restart IRLM, DB2 automatically starts IRLM.

## Monitoring threads

The DB2 command DISPLAY THREAD displays current information about the status of threads, including information about:
- Threads that are processing locally
- Threads that are processing distributed requests
- Stored procedures or user-defined functions if the thread is executing one of those
- Parallel tasks

Threads can be active or inactive:
- An active allied thread is a thread that is connected to DB2 from TSO, BATCH, IMS, CICS, CAF or RRSAF.
- An active database access thread is one connected through a network with another system and performing work on behalf of that system.
- An inactive database access thread is one that is connected through a network to another system and is idle, waiting for a new unit of work to begin from that system. Inactive threads hold no database locks.

The output of the command DISPLAY THREAD can also indicate that a system quiesce is in effect as a result of the ARCHIVE LOG command. For more information, see “Archiving the log” on page 337.

The command DISPLAY THREAD allows you to select which type of information you wish to include in the display using one or more of the following standards:
- Active, indoubt, postponed abort, or inactive threads
- Allied threads associated with the address spaces whose connection-names are specified
- Allied threads
- Distributed threads
- Distributed threads associated with a specific remote location
- Detailed information about connections with remote locations
- A specific logical unit of work ID (LUWID).

The information returned by the DISPLAY THREAD command reflects a dynamic status. By the time the information is displayed, it is possible that the status could have changed. Moreover, the information is consistent only within one address space and is not necessarily consistent across all address spaces.

To use the TYPE, LOCATION, DETAIL, and LUWID keywords, you must have SYSOPR authority or higher. For detailed information, see Chapter 2 of *DB2 Command Reference*.

## Display thread output

DISPLAY THREAD shows active and inactive threads in a format like this:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS:
DSNV402I - ACTIVE THREADS:
  NAME   ST  A     REQ       ID      AUTHID    PLAN     ASID    TOKEN
conn-name s  *   req-ct   corr-id    auth-id   pname    asid    token
conn-name s  *   req-ct   corr-id    auth-id   pname    asid    token
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - module_name '-DISPLAY THREAD' NORMAL COMPLETION
```

DISPLAY THREAD shows indoubt threads in a format like this:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV406I - INDOUBT THREADS -
COORDINATOR               STATUS      RESET  URID         AUTHID
coordinator-name          status      yes/no urid         authid
DISPLAY INDOUBT REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

DISPLAY THREAD shows postponed aborted threads in a format like this:

```
DSNV401I ! DISPLAY THREAD REPORT FOLLOWS -
DSNV431I ! POSTPONED ABORT THREADS -
COORDINATOR               STATUS      RESET URID         AUTHID
coordinator-name          ABORT-P           urid         authid
DISPLAY POSTPONED ABORT REPORT COMPLETE
DSN9022I ! DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

More information about how to interpret this output can be found in the sections describing the individual connections and in the description of message DSNV408I in Part 2 of *DB2 Messages and Codes*.

## Controlling TSO connections

MVS provides no commands for controlling or monitoring a connection to DB2. The connection is monitored instead by the DB2 command -DISPLAY THREAD, which displays information about connections to DB2 (from other subsystems as well as from MVS).

The command is generally entered from an MVS console or an administrator's TSO session. See “Monitoring threads” on page 283 for more examples of its use.

## Connecting to DB2 from TSO

The MVS operator is not involved in starting and stopping TSO connections. Those connections are made through the DSN command processor, which is invoked either

- Explicitly, by the DSN command, or
- Implicitly, through DB2I (DB2 Interactive)

When a DSN session is active, you can enter DSN subcommands, DB2 commands, and TSO commands, as described under “Running TSO application programs” on page 259.

The DSN command can be given in the foreground or background, when running under the TSO terminal monitor program (TMP). The full syntax of the command is:

```
DSN SYSTEM (subsystemid) RETRY (n1) TEST (n2)
```

The parameters are optional, and have the following meanings:

*subsystemid*
Is the subsystem ID of the DB2 subsystem to be connected

*n1* Is the number of times to attempt the connection if DB2 is not running (one attempt every 30 seconds)

*n2* Is the DSN tracing system control that can be used if a problem is suspected

For example, this invokes a DSN session, requesting 5 retries at 30-second intervals:

```
DSN SYSTEM (DB2) RETRY (5)
```

DB2I invokes a DSN session when you select any of these operations:

- SQL statements using SPUFI
- DCLGEN
- BIND/REBIND/FREE
- RUN
- DB2 commands
- Program preparation and execution

In carrying out those operations, the DB2I panels invoke CLISTs, which start the DSN session and invoke appropriate subcommands.

## Monitoring TSO and CAF connections

To display information about connections that use the TSO attach facility and call attach facility (CAF), issue the command DISPLAY THREAD. Table 63 summarizes how DISPLAY THREAD output differs for a TSO online application, a TSO batch application, a QMF session, and a call attach facility application.

*Table 63. Differences in display thread information for TSO and batch*

| **Connection** | **Name** | **AUTHID** | **Corr-ID**[1] | **Plan**[1] |
|---|---|---|---|---|
| DSN (TSO Online) | TSO | Logon ID | Logon ID | RUN .. Plan(x) |
| DSN (TSO Batch) | BATCH | Job USER= | Job Name | RUN .. Plan(x) |
| QMF | DB2CALL | Logon ID | Logon ID | 'QMFvr0' |
| CAF | DB2CALL | Logon ID | Logon ID | OPEN parm |

**Note:**

1. After the application has connected to DB2 but before a plan has been allocated, this field is blank.

The name of the connection can have one of the following values:

**Name** **Connection to**

**TSO** Program running in TSO foreground

**BATCH**
Program running in TSO background

**DB2CALL**
Program using the call attachment facility and running in the same address space as a program using the TSO attachment facility

The correlation ID, *corr-id*, is either the foreground authorization ID or the background job name. For a complete description of the -DISPLAY THREAD status information displayed, see the description of message DSNV404I in Part 2 of *DB2 Messages and Codes*.

The following command displays information about TSO and CAF threads, including those processing requests to or from remote locations:

```
-DISPLAY THREAD(BATCH,TSO,DB2CALL)
```

```
DSNV401I = DISPLAY THREAD REPORT FOLLOWS -
DSNV402I = ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
1 BATCH    T  *  2997 TEP2         SYSADM   DSNTEP41 0019 18818
2 BATCH    RA *  1246 BINETEP2     SYSADM   DSNTEP44 0022 20556
V445-DB2NET.LUND1.AB0C8FB44C4D=20556 ACCESSING DATA FOR SAN_JOSE
3 TSO      T       12 SYSADM       SYSADM   DSNESPRR 0028  5570
4 DB2CALL  T  * 18472 CAFCOB2      SYSADM   CAFCOB2  001A 24979
5 BATCH    T  *     1 PUPPY        SYSADM   DSNTEP51 0025 20499
6          PT *   641 PUPPY        SYSADM   DSNTEP51 002D 20500
7          PT *   592 PUPPY        SYSADM   DSNTEP51 002D 20501
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I = DSNVDT '-DIS THREAD' NORMAL COMPLETION
```

**Key:**

**1** This is a TSO batch application.

**2** This is a TSO batch application running at a remote location and accessing tables at this location.

**3** This is a TSO online application.

**4** This is a call attachment facility application.

**5** This is an originating thread for a TSO batch application.

**6** This is a parallel thread for the originating TSO batch application thread.

**7** This is a parallel thread for the originating TSO batch application thread.

*Figure 24. Display thread showing TSO and CAF connections*

Detailed information for assisting the console operator in identifying threads involved in distributed processing can be found in “Monitoring threads” on page 283.

## Disconnecting from DB2 while under TSO

The connection to DB2 ends, and the thread is terminated, when:

- You enter the END subcommand.
- You enter DSN again. (A new connection is established immediately.)
- You enter the CANCEL THREAD command.
- You press the attention key (PA1).
- Any of the following operations end:
  - SQL statements using SPUFI
  - DCLGEN
  - BIND/REBIND/FREE
  - RUN
- You are using any of the above operations and you enter END or RETURN.

***A simple session:*** For example, the following command and subcommands establish a connection to DB2, run a program, and terminate the connection:

TSO displays:

```
READY
```

You enter:

```
DSN SYSTEM (DSN)
```

DSN displays:

```
DSN
```

You enter:

```
RUN PROGRAM (MYPROG)
```

DSN displays:

```
DSN
```

You enter:

```
END
```

TSO displays:

```
READY
```

## Controlling CICS connections

The following CICS attachment facility commands can be entered from a CICS terminal to control and monitor connections between CICS and DB2:

**DSNC DISCONNECT**
Terminates threads using a specific DB2 plan

**DSNC DISPLAY**
Displays thread information or statistics

**DSNC MODIFY**
Modifies the maximum number of threads for a transaction or group, or the DFHDCT DESTID entry associated with the RCT ERRDEST parameter

**DSNC STOP**
Disconnects CICS from DB2

**DSNC STRT**
Starts the CICS attachment facility

CICS command responses are sent to the terminal from which the corresponding command was entered, unless the DSNC DISPLAY command or a DB2 command specified an alternative destination. The DSNC STOP and DSNC STRT commands cause the output to be sent to the error message transient data queue defined in the DSNCRCT TYPE=INIT macro. For details on specifying alternate destinations for output, see the descriptions of the DB2 command or the DSNC command in Chapter 2 of *DB2 Command Reference*.

Authorization for the DSNC transaction code is controlled through use of:

- The AUTH= parameter on the DSNCRCT macro
- The EXTSEC= and TRANSEC= parameters on the CICS transaction entry for DSNC
- The DB2 SYSOPR authority, which a user must have in order to use DB2 commands

For details on the DSNCRCT macro, see Part 2 of *DB2 Installation Guide*. For details on the CICS transaction entry parameters, see *CICS for MVS/ESA Resource Definition Guide*. For details on the DB2 SYSOPR authority, see “Chapter 10. Controlling access to DB2 objects” on page 103.

## Connecting from CICS

A connection to DB2 can be started or restarted at any time after CICS initialization. The CICS attachment facility is a set of modules DB2 provides that are loaded into the CICS address space. Use the following command to start the attachment facility:

```
DSNC STRT xx,ssid
```

*x* or *xx* names a particular resource control table suffix (DSNCRCT*x* or DSN2CT*xx*). You can also specify a DB2 subsystem ID (*ssid*) on the command. This overrides the subsystem ID specified in the CICS INITPARM or DSNCRCT TYPE=INIT macro.

You can also start the attachment facility automatically at CICS initialization using a program list table (PLT). For details, see Part 2 of *DB2 Installation Guide*.

### Messages

For information about messages that appear during connection, see Part 2 of *DB2 Messages and Codes*. Those messages begin with “DSN2”.

### Restarting CICS

One function of the CICS attachment facility is to keep data in synchronization between the two systems. If DB2 completes phase 1 but does not start phase 2 of the commit process, the units of recovery being committed are termed *indoubt*. An indoubt unit of recovery might occur if DB2 terminates abnormally after completing phase 1 of the commit process. CICS might commit or roll back work without DB2's knowledge.

DB2 cannot resolve those indoubt units of recovery (that is, commit or roll back the changes made to DB2 resources) until the connection to CICS is restarted. This means that CICS should always be auto-started (START=AUTO in the DFHSIT table) to get all necessary information for indoubt thread resolution available from its log. Avoid cold starting. The START option can be specified in the DFHSIT table, as described in *CICS for MVS/ESA Resource Definition Guide*.

In releases after CICS 4.1, the CICS attachment facility enables the INDOUBTWAIT function to resolve indoubt units of recovery automatically. See *CICS for MVS/ESA Customization Guide* for more information.

If there are CICS requests active in DB2 when a DB2 connection terminates, the corresponding CICS tasks might remain suspended even after CICS is reconnected to DB2. You should purge those tasks from CICS using a CICS-supplied transaction such as:

```
CEMT SET TASK(nn) FORCE
```

See *CICS for MVS/ESA CICS-Supplied Transactions* for more information on CICS-supplied transactions.

If any unit of work is indoubt when the failure occurs, the CICS attachment facility automatically attempts to resolve the unit of work when CICS is reconnected to DB2. Under some circumstances, however, CICS cannot resolve indoubt units of recovery. When this happens, message DSN2001I, DSN2034I, DSN2035I, or DSN2036I is sent to the user-named CICS destination that is specified in the resource control table (RCT). You must recover manually these indoubt units of recovery (see “Recovering indoubt units of recovery manually” on page 289 for more information).

### Displaying indoubt units of recovery

To display a list of indoubt units of recovery, give the command:

```
-DISPLAY THREAD (connection-name) TYPE (INDOUBT)
```

The command produces messages similar to these:

```
DSNV407I -STR INDOUBT THREADS - 480
COORDINATOR               STATUS      RESET URID           AUTHID
CICS41                    INDOUBT         00019B8ADE9E   ADMF001
 V449-HAS NID= CICS41.AACC9B739F125184 AND ID=GT00LE39
DISPLAY INDOUBT REPORT COMPLETE
DSN9022I -STR DSNVDT '-DIS THD' NORMAL COMPLETION
```

For an explanation of the list displayed, see the description of message DSNV408I in Part 2 of *DB2 Messages and Codes*.

### Recovering indoubt units of recovery manually

To recover an indoubt unit of recovery, issue the following command:

```
-RECOVER INDOUBT (connection-name) ACTION (COMMIT) ID (correlation-id)
or
-RECOVER INDOUBT (connection-name) ACTION (ABORT) ID (correlation-id)
```

The default value for *connection-name* is the connection name from which you entered the command. *Correlation-id* is the correlation ID of the thread to be recovered. It can be determined by issuing the command DISPLAY THREAD. Your choice for the ACTION parameter tells whether to commit or roll back the associated unit of recovery. For more details, see “Resolving indoubt units of recovery” on page 363.

The following messages can occur after using the RECOVER command:

```
DSNV414I - THREAD correlation-id COMMIT SCHEDULED
or
DSNV415I - THREAD correlation-id ABORT SCHEDULED
```

For more information about manually resolving indoubt units of recovery, see “Manually recovering CICS indoubt units of recovery” on page 419. For information on the two-phase commit process, as well as indoubt units of recovery, see “Consistency with other systems” on page 359.

### Displaying postponed units of recovery

To display a list of postponed units of recovery, issue the command:

```
-DISPLAY THREAD (connection-name) TYPE (POSTPONED)
```

The command produces messages similar to these:

```
DSNV431I -POSTPONED ABORT THREADS - 480
COORDINATOR               STATUS      RESET URID           AUTHID
CICS41                    P-ABORT           00019B8ADE9E   ADMF001
V449-HAS NID= CICS41.AACC9B739F125184 AND ID=GT00LE39
DISPLAY POSTPONED ABORT REPORT COMPLETE
DSN9022I -STR DSNVDT '-DIS THD' NORMAL COMPLETION
```

For an explanation of the list displayed, see the description of message DSNV408I in Part 2 of *DB2 Messages and Codes*.

## Controlling CICS application connections

This section describes how CICS threads are defined, how you can monitor those threads, and ways you can disconnect those threads.

## Defining CICS threads

Every CICS transaction that accesses DB2 requires a thread to service the DB2 requests. Each thread uses one MVS subtask to execute DB2 code for the CICS application.

When the DSNC STRT command is processed, a limited number of subtasks are attached and connected to DB2 as specified in the resource control table (RCT). Additional subtasks can be created and connected during execution.

Threads are created at the first DB2 request from the application if there is not one already available for the specific DB2 plan.

The THRDS parameter for an RCT entry establishes the following:

- The number of MVS subtasks to start when the attachment facility comes up
- The number of protected threads for the entry

Both of these numbers have an impact on performance, as described in “Recommendations for RCT definitions” on page 637. For more information on specifying the THRDS parameter, see Part 2 of *DB2 Installation Guide*.

At any time during execution, thread subtasks can be created. If the following message is displayed:

```
DSN2017I ATTACHMENT OF A THREAD SUBTASK FAILED
```

it could mean that:

- The maximum allowable number of threads specified was reached. The RCT parameter, THRDMAX, specifies the maximum allowable number of threads; when THRDMAX-2 is reached, the attachment facility begins to purge unused subtasks.
- Not enough storage space was provided for subtask creation. See Part 2 of *DB2 Installation Guide* for more information about how to define storage for subtask creation.

## Monitoring the threads

No operator intervention is required for connecting applications; CICS handles the threads dynamically. You can monitor threads using CICS attachment facility commands or DB2 commands.

***Using CICS attachment facility commands:*** Any authorized CICS user can monitor the threads and change the connection parameters as needed. Operators can use the following CICS attachment facility commands to monitor the threads:

```
DSNC DISPLAY PLAN plan-name destination
or
DSNC DISPLAY TRANSACTION transaction-id destination
```

These commands display the threads that the resource or transaction is using. The following information is provided for each created thread:

- Authorization ID for the plan associated with the transaction (8 characters).
- PLAN/TRAN name (8 characters).
- A or I (1 character).

  If **A** is displayed, the thread is within a unit of work. If **I** is displayed, the thread is waiting for a unit of work, and the authorization ID is blank.

The following CICS attachment facility command is used to monitor the RCT:

```
DSNC DISPLAY STATISTICS destination
```

This is an example of the output for the DSNC DISPLAY STATISTICS command:

```
DSN2014I   STATISTICS REPORT FOR 'DSNCRCTC' FOLLOWS
                                                         -----COMMITS-----
TRAN  PLAN         CALLS   AUTHS     W/P HIGH   ABORTS   1-PHASE   2-PHASE
DSNC                   1       1       1    1       0         0         0
POOL  POOL             0       0       0    0       0         0         0
XC01  DSNXC01         22       1      11    2       0         7         5
XC02  DSNXC02          0       0       0    0       0         0         0
XA81  DSNA81           0       0       0    0       0         0         0
XCD4  DSNCED4          0       0       0    0       0         0         0
XP03  DSNTP03          1       1       0    1       0         1         0
XA20  DSNTA20          1       1       0    1       0         0         1
XA88  ********         0       0       0    0       0         0         0
DSN2020I   THE DISPLAY COMMAND IS COMPLETE
```

The DSNC DISPLAY STATISTICS command displays the following information for each entry in the RCT:

**Item** **Description**

**TRAN** Transaction name. For group entries, this is the name of the first transaction defined in the group. DSNC shows the statistics for the TYPE=COMD RCT entry. POOL shows statistics for the TYPE=POOL entry, unless the TYPE=POOL entry contains the parameter TXID=x.

**PLAN** The plan name associated with this entry. Eight asterisks in this field indicates that this transaction is using dynamic plan allocation. The command processor transaction DSNC does not have a plan associated with it because it uses a command processor.

**CALLS**
The total number of SQL statements issued by transactions associated with this entry.

**AUTHS**
The total number of sign-on invocations for transactions associated with this entry. A sign-on does not indicate whether a new thread is created or an existing thread is reused. If the thread is reused, a sign-on occurs only if the authorization ID or transaction ID has changed.

**W/P** The number of times that all available threads for this entry were busy. This value depends on the value of TWAIT for the entry.

If TWAIT was set to POOL in the RCT, W/P indicates the number of times the transaction overflowed to the pool. An overflow to the pool shows up in the transaction statistics only and is not reflected in the pool statistics.

If TWAIT was set to YES, this reflects the number of times that the thread both had to wait, and could not attach a new subtask (number of started tasks has reached THRDA).

The only time W/P is updated for the pool is when a transaction had to wait for a pool thread and a new subtask could not be attached for the pool. The W/P statistic is useful for determining if there are enough threads defined for the entry.

Under normal conditions, you can see a W/P value greater than 0 when the HIGH value has not exceeded the THRDA value. A W/P value greater than 0 occurs because the thread release is asynchronous to the new work coming in and the current high count is decremented before the thread has been marked available when there is no work on the queue.

**HIGH** The maximum number of threads required by transactions associated with this entry at any time since the connection was started. This number includes the transactions that were forced to wait or diverted to the pool. It provides a basis for setting the maximum number of threads for the entry.

**ABORTS**
The total number of units of recovery which were rolled back. It includes both abends and SYNCPOINT ROLLBACKS, including SYNCPOINT ROLLBACKS generated by -911 SQL codes.

**COMMITS**
One of the following two fields is incremented each time a DB2 transaction associated with this entry has a real or implied (such as EOT) syncpoint. Units of recovery that do not process SQL calls are not reflected here.

**1-PHASE**
The total number of single phase commits for transactions associated with this entry. This total does not include any 2-phase commits (see the explanation for 2-PHASE below). This total does include read-only commits as well as single phase commits for units of recovery which have performed updates. A 2-phase commit is needed only when CICS is the recovery coordinator for more than one resource manager.

**2-PHASE**
The total number of 2-phase commits for transactions associated with this entry. This number does not include 1-phase commit transactions.

***Using the DB2 command DISPLAY THREAD:*** The DB2 command DISPLAY THREAD can be used to display CICS attachment facility threads. Some of this information differs depending on whether the connection to CICS is under a control TCB or a transaction TCB.

Table 64 summarizes these differences.

*Table 64. Differences in DISPLAY THREAD information by CICS TCB type*

| Connection | Name | AUTHID[2] | ID[1,2] | Plan[1,2] |
|---|---|---|---|---|
| Control TCB | APPLID | N/A | N/A | N/A |
| Transaction TCB | APPLID | AUTH= on RCT | THRD#TRANID | PLAN= or PLNPGME= on RCT |

**Notes:**

1. After the application has connected to DB2 but before sign-on processing has completed, this field is blank.
2. After sign-on processing has completed but before a plan has been allocated, this field is blank.

The following command displays information about CICS threads, including those accessing data at remote locations:

```
-DISPLAY THREAD(applid)
```

```
DSNV401I = DISPLAY THREAD REPORT FOLLOWS -
DSNV402I = ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
1 CICS41   N        3            SYSADM           001B    0
2 CICS41   T  *     9 PC00DSNC     SYSADM           001B   89
3 CICS41   N        5 PT01XP11     SYSADM           001B    0
4 CICS41   N        0                                001B    0
           CICS41   N        0                                001B    0
5 CICS41   T        4 GT00XP05     SYSADM   TESTP05  001B  171
           CICS41   N        0                                001B    0
           CICS41   N        0                                001B    0
           CICS41   N        0                                001B    0
           CICS41   N        0                                001B    0
           CICS41   N        0                                001B    0
6 CICS41   TR       4 GT01XP05     SYSADM   TESTP05  001B  235
           V444-DB2NET.LUND0.AA8007132465=16 ACCESSING DATA AT
           V446-SAN_JOSE:LUND1
7 CICS41   T  *     3 GC00DSNC     SYSADM           001B  254
DISPLAY ACTIVE REPORT COMPLETE
```

**Key:**

**1** This is the control TCB.

**2** This is a pool connection (first letter ″P″) space executing a command (second letter ″C″). ″*″ in the status column indicates that the thread is processing in DB2.

**3** This is a pool connection that last ran transaction XP11 but the thread has terminated.

**4** This is a connection created by THRDS>0 but has not been used yet.

**5** This is an active entry connection (first letter ″G″) in the CICS address space running transaction XP05.

**6** This is an active entry connection running transaction XP05 with remote activity.

**7** This is an active TYPE=COMD connection executing a command. ″*″ in the status column indicates that the thread is processing in DB2.

*Figure 25. DISPLAY THREAD output showing CICS connections*

## Changing connection parameters

You can use the DSNC MODIFY command to change:

- The destination entry for sending unsolicited messages, as given in the RCT.

  `DSNC MODIFY DESTINATION` *`old new`*
- The actual maximum number of threads for the named transaction (THRDA).

  `DSNC MODIFY TRANSACTION` *`transaction-id integer`*

  The upper limit for this change is the THRDM specified in RCT. *integer* is a new maximum value.

## Disconnecting applications

There is no way to disconnect a particular CICS transaction from DB2 without abending the transaction. Two ways to disconnect an application are described here:

- The DB2 command CANCEL THREAD can be used to cancel a particular thread. CANCEL THREAD requires that you know the *token* for any thread you want to cancel. Enter the following command to cancel the thread identified by the token indicated in the display output:

  ```
  -CANCEL THREAD(46)
  ```

  When you issue CANCEL THREAD for a thread, that thread is scheduled to be terminated in DB2.
- The command DSNC DISCONNECT terminates the threads allocated to a plan ID, but it does not prevent new threads from being created. This command frees DB2 resources shared by the CICS transactions and allows exclusive access to them for special-purpose processes such as utilities or data definition statements.

  To guarantee that no new threads are created for a plan ID, all CICS-related transactions must be disabled before users enter DSNC DISCONNECT. All transactions in a group have the same plan ID, unless dynamic plan selection is specified in the RCT entry for the group. If dynamic plan selection is used, the plan associated with a transaction is determined at execution time.

  The thread is not canceled until the application releases it for reuse, either at SYNCPOINT or end-of-task.

## Disconnecting from CICS

This section describes how to do both an orderly and forced disconnection of the attachment to CICS.

### Orderly termination

It is recommended that you do orderly termination whenever possible. An orderly termination of the connection allows each CICS transaction to terminate before thread subtasks are detached. This means there should be no indoubt units of recovery at reconnection time. An orderly termination occurs when you:

- Enter the DSNC STOP QUIESCE command. CICS and DB2 remain active.
- Enter the CICS command CEMT PERFORM SHUTDOWN, and the CICS attachment facility is also named to shut down during program list table (PLT) processing. DB2 remains active. For information about the CEMT PERFORM SHUTDOWN command, see *CICS for MVS/ESA CICS-Supplied Transactions*.
- Enter the DB2 command STOP DB2 MODE (QUIESCE). CICS remains active.
- Enter the DB2 command CANCEL THREAD. The thread is abended.

The following example stops the DB2 subsystem (QUIESCE), allows the currently identified tasks to continue normal execution, and does not allow new tasks to identify themselves to DB2:

```
-STOP DB2 MODE (QUIESCE)
```

This message appears when the stop process starts and frees the entering terminal (option QUIESCE):

```
DSNC012I THE ATTACHMENT FACILITY STOP QUIESCE IS PROCEEDING
```

When the stop process ends and the connection is terminated, this message is added to the output from the CICS job:

```
DSNC025I THE ATTACHMENT FACILITY IS INACTIVE
```

### Forced termination

Although it is not recommended, there might be times when it is necessary to force the connection to end. A forced termination of the connection can abend CICS transactions connected to DB2. Therefore, indoubt units of recovery can exist at reconnect. A forced termination occurs in the following situations:

- You enter the DSNC STOP FORCE command. This command waits 15 seconds before detaching the thread subtasks, and, in some cases, can achieve an orderly termination. DB2 and CICS remain active.
- You enter the CICS command CEMT PERFORM SHUTDOWN IMMEDIATE. For information about this command, see *CICS for MVS/ESA CICS-Supplied Transactions*. DB2 remains active.
- You enter the DB2 command STOP DB2 MODE (FORCE). CICS remains active.
- A DB2 abend occurs. CICS remains active.
- CICS abend occurs. DB2 remains active.
- STOP is issued to the DB2 or CICS attachment facility, and the CICS transaction overflows to the pool. The transaction issues an intermediate commit. The thread is terminated at commit time, and further DB2 access is not allowed.

This message appears when the stop process starts and frees the entering terminal (option FORCE):

```
DSNC022I THE ATTACHMENT FACILITY STOP FORCE IS PROCEEDING
```

When the stop process ends and the connection is terminated, this message is added to the output from the CICS job:

```
DSNC025I THE ATTACHMENT FACILITY IS INACTIVE
```

## Controlling IMS connections

IMS provides these operator commands for controlling and monitoring the connection to DB2:

**/START SUBSYS**
Connects the IMS control region to a DB2 subsystem

**/TRACE**
Controls the IMS trace

**/DISPLAY SUBSYS**
Displays connection status and thread activity

**/DISPLAY OASN SUBSYS**
Displays outstanding units of recovery

**/CHANGE SUBSYS**
Deletes an indoubt unit of recovery from IMS

**/STOP SUBSYS**
Disconnects IMS from a DB2 subsystem

For more information about those commands, please refer, in the DB2 library, to Chapter 2 of *DB2 Command Reference* or, in the IMS library, to *IMS Command Reference*.

IMS command responses are sent to the terminal from which the corresponding command was entered. Authorization to enter IMS commands is based on IMS security.

### Connecting to the IMS control region

IMS makes one connection to its control region from each DB2 subsystem. IMS can make the connection either:

- Automatically during IMS cold start initialization or at warm start of IMS if a DB2 connection was active when IMS is shut down

- In response to the command /START SUBSYS *ssid*, where *ssid* is the DB2 subsystem identifier

  The command causes the following message to be displayed at the logical terminal (LTERM):

  ```
  DFS058   START COMMAND COMPLETED
  ```

  The message is issued regardless of whether DB2 is active and does not imply that the connection is established.

The order of starting IMS and DB2 is not vital. If IMS is started first, then when DB2 comes up, it posts the control region modify task, and IMS again tries to reconnect.

If DB2 is stopped by the STOP DB2 command, the /STOP SUBSYS command, or a DB2 abend, then IMS cannot reconnect automatically. You must make the connection by using the /START command.

The following messages can be produced when IMS attempts to connect a DB2 subsystem:

- If DB2 is active, these messages are sent:
  - To the MVS console:

    ```
    DFS3613I ESS TCB INITIALIZATION COMPLETE
    ```
  - To the IMS master terminal:

    ```
    DSNM001I IMS/VS imsid CONNECTED TO SUBSYSTEM ssnm
    ```
- If DB2 is not active, this message is sent to the master terminal:

  ```
  DSNM003I IMS/VS imsid FAILED TO CONNECT TO SUBSYSTEM ssnm
           RC=00   imsid
  ```

  *imsid* is the IMS connection name. RC=00 means that a notify request has been queued. When DB2 starts, IMS is also notified.

  No message goes to the MVS console.

## Thread attachment

Execution of the program’s first SQL statement causes the IMS attachment facility to create a thread and allocate a plan, whose name is associated with the IMS application program module name. DB2 sets up control blocks for the thread and loads the plan.

***Using the DB2 command DISPLAY THREAD:*** The DB2 command DISPLAY THREAD can be used to display IMS attachment facility threads.

DISPLAY THREAD output for DB2 connections to IMS differs depending on whether DB2 is connected to a DL/I batch program, a control region, a message-driven program, or a nonmessage-driven program. Table 65 summarizes these differences.

*Table 65. Differences in DISPLAY THREAD information for IMS connections*

| Connection | Name | AUTHID[2] | ID[1,2] | Plan[1,2] |
|---|---|---|---|---|
| DL/I Batch | DDITV02 statement | JOBUSER= | Job Name | DDITV02 statement |
| Control Region | IMSID | N/A | N/A | N/A |
| Message Driven | IMSID | Signon ID or ltermid | PST+ PSB | RTT or program |

Table 65. Differences in DISPLAY THREAD information for IMS connections (continued)

| Connection | Name | AUTHID[2] | ID[1,2] | Plan[1,2] |
|---|---|---|---|---|
| Non-message Driven | IMSID | AXBUSER or PSBNAME | PST+ PSB | RTT or program |

**Notes:**

1. After the application has connected to DB2 but before sign-on processing has completed, this field is blank.
2. After sign-on processing has completed but before a plan has been allocated, this field is blank.

The following command displays information about IMS threads, including those accessing data at remote locations:

```
-DISPLAY THREAD(imsid)
```

```
 DSNV401I -STR DISPLAY THREAD REPORT FOLLOWS -
 DSNV402I -STR ACTIVE THREADS -
 NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
1 SYS3     T  *     3 0002BMP255   ADMF001  PROGHR1  0019    99
           SYS3     T  *     4 0001BMP255   ADMF001  PROGHR2  0018    97
2 SYS3     N        5             SYSADM            0065     0
 DISPLAY ACTIVE REPORT COMPLETE
 DSN9022I -STR DSNVDT '-DIS THD' NORMAL COMPLETION
```

**Key:**

1 This is a message-driven BMP.

2 This thread has completed sign-on processing, but a DB2 plan has not been allocated.

Figure 26. DISPLAY THREAD output showing IMS connections

## Thread termination

When an application terminates, IMS invokes an exit routine to disconnect the application from DB2. There is no way to terminate a thread without abending the IMS application with which it is associated. Two ways of terminating an IMS application are described here:

- Termination of the application

  The IMS commands /STOP REGION *reg#* ABDUMP or /STOP REGION *reg#* CANCEL can be used to terminate an application running in an online environment. For an application running in the DL/I batch environment, the MVS command CANCEL can be used. See *IMS Command Reference* for more information on terminating IMS applications.

- Use of the DB2 command CANCEL THREAD

  CANCEL THREAD can be used to cancel a particular thread or set of threads. CANCEL THREAD requires that you know the *token* for any thread you want to cancel. Enter the following command to cancel the thread identified by a token in the display output:

  ```
  -CANCEL THREAD(46)
  ```

  When you issue CANCEL THREAD for a thread, that thread is scheduled to be terminated in DB2.

## Displaying indoubt units of recovery

**General-use Programming Interface**

One function of the thread connecting DB2 to IMS is to keep data in synchronization between the two systems. If the application program requires it, a change to IMS data must also be made to DB2 data. If DB2 abends while connected to IMS, it is possible for IMS to commit or back out work without DB2 being aware of it. When DB2 restarts, that work is termed *indoubt*. Typically, some decision must be made about the status of the work.

The subject of indoubt units of recovery is treated in detail in “Chapter 19. Restarting DB2 after termination” on page 347.

To display a list of indoubt units of recovery, give the command:

```
-DISPLAY THREAD (imsid) TYPE (INDOUBT)
```

The command produces messages similar to these:

```
DSNV401I -STR DISPLAY THREAD REPORT FOLLOWS -
DSNV406I -STR POSTPONED ABORTT THREADS - 920
COORDINATOR              STATUS      RESET URID         AUTHID
SYS3                     P-ABORT           00017854FF6B ADMF001
V449-HAS NID= SYS3.400000000 AND ID= 0001BMP255
BATCH                    P-ABORT           00017854A8A0 ADMF001
V449-HAS NID= DSN:0001.0 AND ID= RUNP10
BATCH                    P-ABORT           00017854AA2E ADMF001
V449-HAS NID= DSN:0002.0 AND ID= RUNP90
BATCH                    P-ABORT           0001785CD711 ADMF001
V449-HAS NID= DSN:0004.0 AND ID= RUNP12
DISPLAY POSTPONED ABORT REPORT COMPLETE
DSN9022I -STR DSNVDT '-DIS THD' NORMAL COMPLETION
```

For an explanation of the list displayed, see the description of message DSNV408I in Part 2 of *DB2 Messages and Codes*.

**End of General-use Programming Interface**

## Recovering indoubt units of recovery

**General-use Programming Interface**

To recover an indoubt unit, issue the following command:

```
-RECOVER INDOUBT (imsid) ACTION (COMMIT) ID (pst#.psbname)
or
-RECOVER INDOUBT (imsid) ACTION (ABORT) ID (pst#.psbname)
```

Here *imsid* is the connection name and *pst#.psbname* is the correlation ID listed by the command DISPLAY THREAD. Your choice of the ACTION parameter tells whether to commit or roll back the associated unit of recovery. For more details, see “Resolving indoubt units of recovery” on page 363.

The following messages can occur after using the RECOVER command:

```
DSNV414I - THREAD pst#.psbname COMMIT SCHEDULED
or
DSNV415I - THREAD pst#.psbname ABORT SCHEDULED
```

**End of General-use Programming Interface**

## Displaying postponed units of recovery

**General-use Programming Interface**

The subject of postponed units of recovery is treated in detail in “Chapter 19. Restarting DB2 after termination” on page 347. This chapter describes the operational steps used to list and recover postponed units in relatively simple cases.

To display a list of postponed units of recovery, give the command:

```
-DISPLAY THREAD (imsid) TYPE (POSTPONED)
```

The command produces messages similar to these:

```
DSNV401I -STR DISPLAY THREAD REPORT FOLLOWS -
DSNV406I -STR POSTPONED ABORTT THREADS - 920
COORDINATOR               STATUS      RESET URID         AUTHID
SYS3                      P-ABORT           00017854FF6B ADMF001
V449-HAS NID= SYS3.400000000 AND ID= 0001BMP255
BATCH                     P-ABORT           00017854A8A0 ADMF001
V449-HAS NID= DSN:0001.0 AND ID= RUNP10
BATCH                     P-ABORT           00017854AA2E ADMF001
V449-HAS NID= DSN:0002.0 AND ID= RUNP90
BATCH                     P-ABORT           0001785CD711 ADMF001
V449-HAS NID= DSN:0004.0 AND ID= RUNP12
DISPLAY POSTPONED ABORT REPORT COMPLETE
DSN9022I -STR DSNVDT '-DIS THD' NORMAL COMPLETION
```

For an explanation of the list displayed, see the description of messages in Part 2 of *DB2 Messages and Codes*.

**End of General-use Programming Interface**

## Duplicate correlation IDs

**General-use Programming Interface**

It is possible for two threads to have the same correlation ID (*pst#.psbname*) if all of these conditions occur:

- Connections have been broken several times.
- Indoubt units of recovery were not recovered.
- Applications were subsequently scheduled in the same region.

To uniquely identify threads which have the same correlation ID (*pst#.psbname*) requires that you be able to identify and understand the network ID (NID). For connections with IMS, you should also be able to identify and understand the IMS originating sequence number (OASN).

The NID is shown in a condensed form on the messages issued by the DB2 DISPLAY THREAD command processor. The IMS subsystem name (*imsid*) is displayed as the *net_node*. The *net_node* is followed by the 8-byte OASN, displayed in hexadecimal format (16 characters), with all leading zeros omitted. The *net_node* and the OASN are separated by a period.

For example, if the *net_node* is IMSA, and the OASN is 0003CA670000006E, the NID is displayed as IMSA.3CA670000006E on the DB2 DISPLAY THREAD command output.

If two threads have the same *corr-id*, use the NID instead of *corr-id* on the RECOVER INDOUBT command. The NID uniquely identifies the work unit.

The OASN is a 4-byte number that represents the number of IMS scheduling since the last IMS cold start. The OASN is occasionally found in an 8-byte format, where the first four bytes contain the scheduling number, and the last four bytes contain the number of IMS sync points (commits) during this schedule. The OASN is part of the NID.

The NID is a 16-byte network ID that originates from IMS. The NID contains the 4-byte IMS subsystem name, followed by four bytes of blanks, followed by the 8-byte version of the OASN. In communications between IMS and DB2, the NID serves as the recovery token.

**End of General-use Programming Interface**

### Resolving residual recovery entries

At given times, IMS builds a list of *residual recovery entries* (RREs). RREs are units of recovery about which DB2 could be in doubt. They arise in several situations:

- If DB2 is not operational, IMS has RREs that cannot be resolved until DB2 is operational. Those are not a problem.
- If DB2 is operational and connected to IMS, and if IMS rolled back the work that DB2 has committed, the IMS attachment facility issues message DSNM005I. If the data in the two systems must be consistent, this is a problem situation. Its resolution is discussed in “Resolution of indoubt units of recovery” on page 414.
- If DB2 is operational and connected to IMS, RREs can still exist, even though no messages have informed you of this problem. The only way to recognize this problem is to issue the IMS /DISPLAY OASN SUBSYS command after the DB2 connection to IMS has been established.

  To display the RRE information, give the command:

  ```
  /DISPLAY OASN SUBSYS subsystem-name
  ```

  To purge the RRE, give one of these commands:

  ```
  /CHANGE SUBSYS subsystem-name RESET
  /CHANGE SUBSYS subsystem-name RESET OASN nnnn
  ```

  where *nnnn* is the originating application sequence number listed in the display. That is the schedule number of the program instance, telling its place in the sequence of invocations of that program since the last cold start of IMS. IMS cannot have two indoubt units of recovery with the same schedule number.

  Those commands reset the status of IMS; they do not result in any communication with DB2.

## Controlling IMS dependent region connections

Controlling IMS dependent region connections involves three activities:

- Connecting from dependent regions
- Monitoring the activity on connections
- Disconnecting from dependent regions

### Connecting from dependent regions

The IMS attachment facility used in the control region is also loaded into dependent regions. A connection is made from each dependent region to DB2. This connection is used to pass SQL statements and to coordinate the commitment of DB2 and IMS work. The following process is used by IMS to initialize and connect.

1. Read the SSM from IMS.PROCLIB.

   A subsystem member can be specified on the dependent region EXEC parameter. If it is not specified, the control region SSM is used. If the region will never connect to DB2, specify a member with no entries to avoid loading the attachment facility.

2. Load the DB2 attachment facility from *prefix*.SDSNLOAD

   For a batch message processing (BMP) program, the load is not done until the application issues its first SQL statement. At that time, IMS attempts to make the connection.

   For a message processing program (MPP) region or IMS Fast Path (IFP) region, the connection is made when the IMS region is initialized, and an IMS transaction is available for scheduling in that region.

   An IMS dependent region establishes two connections to DB2: a region connection and an application connection, which occurs at execution of the first SQL statement.

If DB2 is not active, or if resources are not available when the first SQL statement is issued from an application program, the action taken depends on the error option specified on the SSM user entry. The options are:

**Option Action**

**R** The appropriate return code is sent to the application, and the SQL code is returned.

**Q** The application is abended. This is a PSTOP transaction type; the input transaction is re-queued for processing and new transactions are queued.

**A** The application is abended. This is a STOP transaction type; the input transaction is discarded and new transactions are not queued.

The region error option can be overridden at the program level via the resource translation table (RTT). See Part 2 of *DB2 Installation Guide* for further details.

## Monitoring the activity on connections

The information under this heading, up to “Disconnecting from dependent regions” on page 303, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

A thread is established from a dependent region when an application makes its first successful DB2 request. Information on connections and the applications currently using them can be displayed by issuing one of these commands:

**From DB2:**
-DISPLAY THREAD (*imsid*)

**From IMS:**
/SSR -DISPLAY THREAD (*imsid*)

Either command produces the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
NAME      ST  A  REQ     ID           AUTHID     PLAN     ASID    TOKEN
conn-name s   *  req-ct  corr-id      auth-id    pname    asid    token
conn-name s   *  req-ct  corr-id      auth-id    pname    asid    token
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

For an explanation of the -DISPLAY THREAD status information displayed, see the description of message DSNV404I in Part 2 of *DB2 Messages and Codes*. More detailed information regarding use of this command and the reports it produces is available in “The command DISPLAY THREAD” on page 312.

IMS provides a display command to monitor the connection to DB2. In addition to showing which program is active on each dependent region connection, the display also shows the LTERM user name and gives the control region connection status. The command is:

```
/DISPLAY SUBSYS subsystem-name
```

The status of the connection between IMS and DB2 is shown as one of the following:

```
CONNECTED
NOT CONNECTED
CONNECT IN PROGRESS
STOPPED
STOP IN PROGRESS
INVALID SUBSYSTEM NAME=name
SUBSYSTEM name NOT DEFINED BUT RECOVERY OUTSTANDING
```

The thread status from each dependent region is:

```
CONN
CONN, ACTIVE (includes LTERM of user)
```

The following four examples show the output that might be generated when an IMS /DISPLAY SUBSYS command is issued.

```
0000 15.49.57           R 45,/DIS SUBSYS NEW
0000 15.49.57           IEE600I REPLY TO 45 IS;/DIS SUBSYS END
0000 15.49.57 JOB   56  DFS000I DSNM003I IMS/TM V1 SYS3 FAILED TO CONNECT TO SUBSYSTEM DSN RC=00  SYS3
0000 15.49.57 JOB   56  DFS000I     SUBSYS   CRC REGID PROGRAM  LTERM    STATUS   SYS3
0000 15.49.57 JOB   56  DFS000I     DSN      :                           NON CONN    SYS3
0000 15.49.57 JOB   56  DFS000I     *83228/154957*   SYS3
0000 15.49.57 JOB   56 *46 DFS996I *IMS READY*  SYS3
```

*Figure 27. Example of output from IMS /DISPLAY SUBSYS processing for a DSN subsystem that is not yet connected. Message DSNM003I is issued by the IMS attachment facility.*

```
0000 15.58.59           R 46,/DIS SUBSYS ALL
0000 15.58.59           IEE600I REPLY TO 46 IS;/DIS SUBSYS ALL
0000 15.59.01 JOB   56  DFS551I MESSAGE REGION MPP1     STARTED ID=0001 TIME=1551 CLASS=001,002,003,004
0000 15.59.01 JOB   56  DFS000I DSNM001I IMS/TM=V1 SYS3 CONNECTED TO SUBSYSTEM DSN   SYS3
0000 15.59.01 JOB   56  DFS000I     SUBSYS   CRC REGID PROGRAM  LTERM    STATUS   SYS3
0000 15.59.01 JOB   56  DFS000I     DSN      :                          CONN    SYS3
0000 15.59.01 JOB   56  DFS000I     *83228/155900*   SYS3
0000 15.59.01 JOB   56 *47 DFS996I *IMS READY*  SYS3
```

*Figure 28. Example of output from IMS /DISPLAY SUBSYS processing for a DSN subsystem that is connected. Message DSNM001I is issued by the IMS attachment facility.*

```
0000 15.59.28           R 47,/STO SUBSYS ALL
0000 15.59.28           IEE600I REPLY TO 47 IS;/STO SUBSYS ALL
0000 15.59.37 JOB   56  DFS058I 15:59:37 STOP COMMAND IN PROGRESS   SYS3
0000 15.59.37 JOB   56 *48 DFS996I *IMS READY*  SYS3
0000 15.59.44           R 48,/DIS SUBSYS ALL
0000 15.59.44           IEE600I REPLY TO 48 IS;/DIS SUBSYS ALL
0000 15.59.45 JOB   56  DFS000I DSNM002I IMS/TM V1 SYS3 DISCONNECTED FROM SUBSYSTEM DSN RC = E.  SYS3
0000 15.59.45 JOB   56  DFS000I     SUBSYS   CRC REGID PROGRAM  LTERM    STATUS   SYS3
0000 15.59.45 JOB   56  DFS000I     DSN      :                           STOPPED    SYS3
0000 15.59.45 JOB   56  DFS000I     *83228/155945*   SYS3
0000 15.59.45 JOB   56 *49 DFS996I *IMS READY*  SYS3
```

*Figure 29. Example of output from IMS /STOP SUBSYS and IMS /DISPLAY SUBSYS commands. The output that is displayed in response to /DISPLAY SUBSYS shows a stopped status. Message DSNM002I is issued by the IMS attachment facility.*

```
0000 16.09.35 JOB   56  R 59,/DIS SUBSYS ALL
0000 16.09.35 JOB   56  IEE600I REPLY TO 59 IS;/DIS SUBSYS ALL
0000 16.09.38 JOB   56  DFS000I     SUBSYS   CRC REGID PROGRAM  LTERM    STATUS   SYS3
0000 16.09.38 JOB   56  DFS000I     DSN      :                           CONN     SYS3
0000 16.09.38 JOB   56  DFS000I                      1                   CONN     SYS3
0000 16.09.38 JOB   56  DFS000I     *83228/160938*   SYS3
0000 16.09.38 JOB   56 *60 DFS996I *IMS READY*  SYS3
0000 16.09.38 JOB   56
```

*Figure 30. Example of output from IMS /DISPLAY SUBSYS processing for a DSN subsystem that is connected and the region ID (1) that is included. Use the REGID(pst#) and the PROGRAM(pstname) values to correlate the output of the command to the LTERM involved.*

### Disconnecting from dependent regions

Usually, IMS master terminal operators do not want to disconnect a dependent region explicitly. However, they might want to change values in the SSM member of IMS.PROCLIB. To do that, they can issue /STOP REGION, update the SSM member, and issue /START REGION.

## Disconnecting from IMS

The connection is ended when either IMS or DB2 terminates. Alternatively, the IMS master terminal operator can explicitly break the connection by entering this command:

```
/STOP SUBSYS subsystem-name
```

That command sends the following message to the terminal that entered it, usually the master terminal operator (MTO):

```
DFS058I STOP COMMAND IN PROGRESS
```

The /START SUBSYS *subsystem-name* command is required to reestablish the connection.

In implicit or explicit disconnect, this message is sent to the IMS master terminal:

```
DSNM002I IMS/TM imsid DISCONNECTED FROM SUBSYSTEM subsystem-name - RC=z
```

That message uses the following reason codes (RC):

**Code** **Meaning**

**A** IMS/TM is terminating normally (for instance, /CHE FREEZE³DUMPQ³PURGE). Connected threads complete.

**B** IMS is abending. Connected threads are rolled back. DB2 data is backed out now; DL/I data is backed out on IMS restart.

**C** DB2 is terminating normally after a -STOP DB2 MODE (QUIESCE) command. Connected threads complete.

**D** DB2 is terminating normally after a -STOP DB2 MODE (FORCE) command, or DB2 is abending. Connected threads are rolled back. DL/I data is backed out now. DB2 data is backed out now if DB2 terminated normally; otherwise, at restart.

**E** IMS is ending the connection because of a /STOP SUBSYS *subsystem-name* command. Connected threads complete.

If an application attempts to access DB2 after the connection ended and before a thread is established, the attempt is handled according to the region error option specification (R, Q, or A).

## Controlling OS/390 RRS connections

**General-use Programming Interface**

Application programs can use the following Recoverable Resource Manager Services attachment facility (RRSAF) functions to control connections to DB2:

**IDENTIFY**
Establishes the task (TCB) as a user of the named DB2 subsystem. When the first task within an address space issues a connection request, the address space is initialized as a user of DB2.

**SIGNON**
Provides a user ID and optionally, one or more secondary authorization IDs to be associated with the connection. Invokes the sign-on exit routine. Optionally, lets a thread join a global transaction. See “Recommendations for application design” on page 648 for more information about global transactions.

**AUTH SIGNON**
Provides a user ID, an ACEE, and optionally, one or more secondary authorization IDs to be associated with the connection. Invokes the sign-on exit.

**CREATE THREAD**
Allocates a plan. If you provide a plan name, DB2 allocates that plan. If you provide a collection name, DB2 allocates a special plan named ?RRSAF and a package list that contains the collection name.

After CREATE THREAD completes, DB2 can execute SQL statements.

**TERMINATE THREAD**
Deallocates the plan.

**TERMINATE IDENTIFY**
Removes the task as a user of DB2. If this is the last or only task in the address space with a DB2 connection, TERMINATE IDENTIFY terminates the address space connection to DB2.

**TRANSLATE**
Returns an SQL code and printable text, in the SQLCA, that describes a DB2 error reason code.

**End of General-use Programming Interface**

For more information on those functions, see Part 6 of *DB2 Application Programming and SQL Guide*.

## Connecting to OS/390 RRS using RRSAF

The information under this heading up to “Controlling connections to remote systems” on page 307 is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

An RRSAF connection can be started or restarted at any time after OS/390 RRS is started. If OS/390 RRS is not started, an IDENTIFY request fails with reason code X'00F30091'.

### Restarting DB2 and OS/390 RRS

If DB2 abnormally terminates but OS/390 RRS remains active, OS/390 RRS might commit or roll back work without DB2's knowledge. In a similar manner, if OS/390 RRS abnormally terminates after DB2 has completed phase 1 of commit processing for an application, then DB2 does not know whether to commit or roll back the work. In either case, when DB2 restarts, that work is termed *indoubt*.

DB2 cannot resolve those indoubt units of recovery (that is, commit or roll back the changes made to DB2 resources) until DB2 restarts with OS/390 RRS.

If any unit of work is indoubt when a failure occurs, DB2 and OS/390 RRS automatically resolve the unit of work when DB2 restarts with OS/390 RRS.

### Displaying indoubt units of recovery

To display a list of indoubt units of recovery, issue the command:

```
-DISPLAY THREAD (RRSAF) TYPE (INDOUBT)
```

The command produces output similar to this:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV406I - INDOUBT THREADS -
COORDINATOR               STATUS      RESET URID          AUTHID
RRSAF                     INDOUBT           00019B8ADE9E   ADMF001
 V449-HAS NID= AD64101C7EED90000000000101010000 AND ID= ST47653RRS
DISPLAY INDOUBT REPORT COMPLETE
DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

For RRSAF connections, a network ID is the OS/390 RRS Unit of Recovery ID (URID) that uniquely identifies a unit of work. An OS/390 RRS URID is a 32 character number. For an explanation of the output, see the description of message DSNV408I in Part 2 of *DB2 Messages and Codes*.

### Recovering indoubt units of recovery manually

Manual recovery of an indoubt unit of recovery might be required if the OS/390 RRS log is lost. When that happens, message DSN3011I is displayed on the MVS console.

To recover an indoubt unit of recovery, issue one of the following commands:

```
-RECOVER INDOUBT (RRSAF) ACTION (COMMIT) ID (correlation-id)
```

or

```
-RECOVER INDOUBT (RRSAF) ACTION (ABORT) ID (correlation-id)
```

*correlation-id* is the correlation ID of the thread to be recovered. You can determine the correlation ID by issuing the command DISPLAY THREAD.

The ACTION parameter indicates whether to commit or roll back the associated unit of recovery. For more details, see “Resolving indoubt units of recovery” on page 363.

If you recover a thread that is part of a global transaction, all threads in the global transaction are recovered.

The following messages can occur when you issue the RECOVER INDOUBT command:

```
DSNV414I - THREAD correlation-id COMMIT SCHEDULED
```

```
DSNV415I - THREAD correlation-id ABORT SCHEDULED
```

If the following DSNV418I message is issued:

```
DSNV418I - RECOVER INDOUBT REJECTED FOR ID=correlation-id
```

the NID option of RECOVER INDOUBT, as shown below, must be used.

```
-RECOVER INDOUBT(RRSAF) ACTION(action) NID(nid)
```

where *nid* is the 32 character field displayed in the DSNV449I message.

For information on the two-phase commit process, as well as indoubt units of recovery, see “Consistency with other systems” on page 359.

### Displaying postponed units of recovery

To display a list of postponed units of recovery, issue the command:

```
-DISPLAY THREAD (RRSAF) TYPE (POSTPONED)
```

The command produces output similar to this:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV406I - POSTPONED ABORT THREADS -
COORDINATOR               STATUS      RESET URID           AUTHID
RRSAF                     P-ABORT           00019B8ADE9E   ADMF001
V449-HAS NID= AD64101C7EED90000000000101010000 AND ID= ST47653RRS
DISPLAY POSTPONED ABORT REPORT COMPLETE
DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

For RRSAF connections, a network ID is the OS/390 RRS Unit of Recovery ID (URID) that uniquely identifies a unit of work. An OS/390 RRS URID is a 32 character number. For an explanation of the output, see the description of message DSNV408I in Part 2 of *DB2 Messages and Codes*.

## Monitoring RRSAF connections

RRSAF allows an application or application monitor to disassociate a DB2 thread from a TCB and later associate the thread with the same or different TCB within the same address space. RRSAF uses the OS/390 RRS Switch Context (CTXSWCH) service to do this. Only authorized programs can execute CTXSWCH.

DB2 stores information in an OS/390 RRS CONTEXT about an RRSAF thread so that DB2 can locate the thread later. An application or application monitor can then invoke CTXSWCH to dissociate the CONTEXT from the current TCB and then associate the CONTEXT with the same TCB or a different TCB.

The following command displays information about RRSAF threads, including those that access data at remote locations:

```
-DISPLAY THREAD(RRSAF)
```

```
 DSNV401I = DISPLAY THREAD REPORT FOLLOWS -
 DSNV402I = ACTIVE THREADS -
 NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
1 RRSAF    T        4 RRSTEST2-111 ADMF001  ?RRSAF   0024    13
2 RRSAF    T        6 RRSCDBTEST01 USRT001  TESTDBD  0024    63
3 RRSAF    DI       3 RRSTEST2-100 USRT002  ?RRSAF   001B    99
4 RRSAF    TR       9 GT01XP05     SYSADM   TESTP05  001B   235
            V444-DB2NET.LUND0.AA8007132465=16 ACCESSING DATA AT
            V446-SAN_JOSE:LUND1
 DISPLAY ACTIVE REPORT COMPLETE
```

**Key:**

1 This is an application that used CREATE THREAD to allocate the special plan used by RRSAF (plan name = ?RRSAF).

2 This is an application that connected to DB2 and allocated a plan with the name TESTDBD.

3 This is an application that is currently not connected to a TCB (shown by status DI).

4 This is an active connection that is running plan TESTP05. The thread is accessing data at a remote site.

*Figure 31. DISPLAY THREAD output showing RRSAF connections*

### Disconnecting applications from DB2

You cannot disconnect an RRSAF transaction from DB2 without abending the transaction. You can use the DB2 command CANCEL THREAD to cancel a particular thread. CANCEL THREAD requires that you know the *token* for any thread that you want to cancel. Issue the command DISPLAY THREAD to obtain the token number, then enter the following command to cancel the thread:

```
-CANCEL THREAD(token)
```

When you issue CANCEL THREAD, DB2 schedules the thread for termination.

## Controlling connections to remote systems

The information under this heading, up to “Using NetView® to monitor errors in the network” on page 323, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

You can control connections to remote systems, which use distributed data, by controlling the threads. Two types of threads are involved with connecting to other systems, *allied threads* and *database access threads*. An allied thread is a thread that is connected locally to your DB2 subsystem, that is from TSO, CICS, IMS, or a stored procedures address space. A database access thread is a thread initiated by a remote DBMS to your DB2 subsystem. The following topics are covered here:

“Starting DDF” on page 308

“Suspending and resuming DDF server activity” on page 308

“Monitoring connections to other systems” on page 309, which describes the use of the following commands:

– DISPLAY DDF

- DISPLAY LOCATION
- DISPLAY THREAD
- CANCEL THREAD
- VARY NET,TERM (VTAM command)

“Monitoring and controlling stored procedures” on page 320
“Using NetView® to monitor errors in the network” on page 323
“Stopping DDF” on page 325

***Related information:*** The following topics in this book contain information about distributed connections:
“Resolving indoubt units of recovery” on page 363
“Failure of a database access thread” on page 446
“Chapter 35. Tuning and monitoring in a distributed environment” on page 857

## Starting DDF

To start the distributed data facility (DDF), if it has not already been started, use the following command:

```
-START DDF
```

When DDF is started and is responsible for indoubt thread resolution with remote partners, one or both of messages DSNL432I and DSNL433I is generated. These messages summarize DDF’s responsibility for indoubt thread resolution with remote partners. See “Chapter 20. Maintaining consistency across multiple systems” on page 359 for information about resolving indoubt threads.

Using the START DDF command requires authority of SYSOPR or higher. The following messages are associated with this command:

```
DSNL003I - DDF IS STARTING

DSNL004I - DDF START COMPLETE LOCATION locname
           LU netname.luname

           GENERICLU netname.gluname
           DOMAIN domain
           TCPPORT tcpport
           RESPORT resport
```

If the distributed data facility has not been properly installed, the START DDF command fails, and message DSN9032I, - REQUESTED FUNCTION IS NOT AVAILABLE is issued. If the distributed data facility has already been started, the START DDF command fails, and message DSNL001I, - DDF IS ALREADY STARTED is issued. Use the DISPLAY DDF command to display the status of DDF.

When you install DB2, you can request that the distributed data facility start automatically when DB2 starts. For information on starting the distributed data facility automatically, see Part 2 of *DB2 Installation Guide*.

## Suspending and resuming DDF server activity

You can use the STOP DDF MODE(SUSPEND) command to suspend DDF server threads temporarily. Suspending DDF server threads frees all resources held by the server threads and lets the following operations complete:

- CREATE
- ALTER
- DROP
- GRANT

- REVOKE

When you issue STOP DDF MODE(SUSPEND), DB2 waits for all active DDF database access threads to become inactive or terminate. Two optional keywords on this command, WAIT and CANCEL, let you control how long DB2 waits and what action DB2 takes after a specified time period. To resume suspended DDF server threads, issue the START DDF command. For more detailed information about the STOP DDF MODE(SUSPEND) command, see Chapter 2 of *DB2 Command Reference*.

## Monitoring connections to other systems

The following DB2 commands give you information about distributed threads:

**DISPLAY DDF**
Displays information about the status and configuration of the distributed data facility (DDF), and about the connections or threads controlled by DDF. For its use, see “The command DISPLAY DDF”.

**DISPLAY LOCATION**
Displays statistics about threads and conversations between remote DB2 subsystem and the local subsystem. For its use, see “The command DISPLAY LOCATION” on page 311.

**DISPLAY THREAD**
Displays information about DB2, distributed subsystem connections, and parallel tasks. For its use, see “The command DISPLAY THREAD” on page 312 .

### The command DISPLAY DDF

The command DISPLAY DDF displays information regarding the status of DDF and, in addition, any information that is displayed when DDF is started, such as the location name, the LU name, the IP address, and domain names. Using the DETAIL keyword provides additional configuration and statistical information.

To issue the DISPLAY DDF command, you must have SYSOPR authority or higher. To issue the command, enter the following:

```
-DISPLAY DDF
```

DB2 returns output similar to this sample when DDF has *not yet* been started:

```
DSNL080I - DSNLTDDF DISPLAY DDF REPORT FOLLOWS-
DSNL081I STATUS=STOPDQ
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I SVL650A            -NONE.SYEC650A    -NONE
DSNL084I IPADDR          TCPPORT RESPORT
DSNL085I -NONE           447     5002
DSNL086I SQL    DOMAIN=-NONE
DSNL086I RESYNC DOMAIN=-NONE
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

DB2 returns output similar to this sample when DDF has been started:

```
DSNL080I - DSNLTDDF DISPLAY DDF REPORT FOLLOWS-
DSNL081I STATUS=STARTD
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I SVL650A            USIBMSY.SYEC650A  -NONE
DSNL084I IPADDR          TCPPORT RESPORT
DSNL085I 8.110.115.106   447     5002
DSNL086I SQL    DOMAIN=v7ic111.svl.ibm.com
DSNL086I RESYNC DOMAIN=v7ic111.svl.ibm.com
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

The DISPLAY DDF command displays the following information:

- The status of the distributed data facility (DDF)
- The location name of DDF defined in the BSDS
- The fully qualified LU name for DDF (that is, the network ID and LUNAME)
- The fully qualified generic LU name for DDF
- The IP address of DDF
- The SQL listener TCP/IP port number
- The two-phase commit resynchronization (resync) listener TCP/IP port number
- The SQL and RESYNC domain names:
  - The SQL domain type accepts inbound SQL requests from remote partners.
  - The RESYNC domain type accepts inbound two-phase commit resynchronization requests.

To use the DETAIL option, enter the following:

```
-DISPLAY DDF DETAIL
```

DB2 returns additional lines of output:

```
DSNL080I - DSNLTDDF DISPLAY DDF REPORT FOLLOWS-
DSNL081I STATUS=STARTD
...
DSNL090I DT=A  CONDBAT=     64 MDBAT=   64
DSNL092I ADBAT=    1 QUEDBAT=      0 IN1DBAT=      0 CONQUED=      0
DSNL093I DSCDBAT=      0 IN2CONS=      0
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

The DISPLAY DDF DETAIL command displays this additional information:

- The DDF thread value (DT) of either **A** or **I**:
  - The DDF is configured with DDF THREADS ACTIVE.
  - The DDF is configured with DDF THREADS INACTIVE.
- The maximum number of inbound connections for database access threads (CONDBAT)
- The maximum number of concurrent active DBATs that could potentially be executing SQL (MDBAT)
- The current number of active database access threads (ADBAT)
- The current number of queued database access threads (QUEDBAT)
- The current number of Type 1 inactive threads (IN1DBAT)
- The current number of connection requests that have been queued and are waiting (CONQUED)
- The current number of disconnected database access threads (DSCDBAT)
- The current number of Type 2 inactive connections (IN2CONS)

For more DISPLAY DDF message information, see Part 2 of *DB2 Messages and Codes*.

The DISPLAY DDF DETAIL command is especially useful because it reflects the presence of new inbound connections that are not reflected by other commands. For example, if DDF is configured with inactive support, as denoted by a DT value of **I** in the DSNL090I message, and if DDF is stopped suspended or the maximum number of active data base access threads has been reached, then new inbound connections are not yet reflected in the DISPLAY THREAD report. However, the presence of these new connections is reflected in the DISPLAY DDF DETAIL report, although specific details regarding the origin of the connections, such as the client system IP address or LU name, are not available until the connections are actually associated with a database access thread.

## The command DISPLAY LOCATION

The command DISPLAY LOCATION displays summary information about connections with other locations and can be used to display detailed information about DB2 system conversations. System conversations are used either for DB2 private protocol access or for supporting functions with DRDA access. Location names, SNA LU names or IP addresses, can be specified, and the DETAIL keyword is supported. To issue the DISPLAY LOCATION command, you must have SYSOPR authority or higher. To issue the command, enter the following:

```
-DISPLAY LOCATION(*)
```

DB2 returns output similar to this sample:

```
DSNL200I - DISPLAY LOCATION REPORT FOLLOWS-
LOCATION         PRDID    LINKNAME        REQUESTERS SERVERS CONVS
USIBMSTODB22     DSN05010 LUND0                    1       0     3
USIBMSTODB23     DSN04010 LUND1                    0       0     0
DRDALOC          SQL03030 124.63.51.17             3       0     3
124.63.51.17     SQL03030 124.63.51.17             0      15    15
DISPLAY LOCATION REPORT COMPLETE
```

You can use an asterisk (*) in place of the end characters of a location name. For example, use -DISPLAY LOCATION(SAN*) to display information about all active connections between your DB2 and a remote location that begins with “SAN”. This includes the number of conversations and the role for each non-system conversation, requester or server.

When DB2 connects with a remote location, information about that location, including LOCATION, PRDID and LINKNAME (LUNAME or IP address), persists in the report even if no active connections exist.

The DISPLAY LOCATION command displays the following types of information for each DBMS that has active threads, except for the local subsystem:

- The location name (or RDB_NAME) of the other connected system. If the RDBNAME is not known, the LOCATION column contains one of the following:
  - A VTAM LU name in this format: ’<luname>’.
  - A dotted decimal IP address in this format: ’nnn.nnn.nnn.nnn’.
- The PRDID, which identifies the database product at the location in the form *nnnvvrrm*:
  - *nnn* - identifies the database product
  - *vv* - product version
  - *rr* - product release
  - *m* - product modification level.
- The corresponding LUNAME or IP address of the system.
- The number of threads at the local system that are requesting data from the remote system.
- The number of threads at the local system that are acting as a server to the remote system.
- The total number of conversations in use between the local system and the remote system. For USIBMSTODB23, in the sample output above, the locations are connected and system conversations have been allocated, but currently there are no active threads between the two sites.

DB2 does not receive a location name from non-DB2 requesting DBMSs that are connected to DB2. In this case, it displays instead the LUNAME of the requesting DBMS, enclosed in less-than (<) and greater-than (>) symbols.

For example, suppose there are two threads at location USIBMSTODB21. One is a distributed access thread from a non-DB2 DBMS, and the other is an allied thread going from USIBMSTODB21 to the non-DB2 DBMS. The DISPLAY LOCATION command issued at USIBMSTODB21 would display output similar to the following:

```
DSNL200I - DISPLAY LOCATION REPORT FOLLOWS -
LOCATION         PRDID    LINKNAME        REQUESTERS SERVERS CONVS
NONDB2DBMS                LUND1                    1       0      1
<LULA>           DSN04010 LULA                     0       1      1
DISPLAY LOCATION REPORT COMPLETE
```

The output below shows the result of a DISPLAY LOCATION(*) command when DB2 is connected to the following DRDA partners:

- DB2A is connected to this DB2, using TCP/IP for DRDA connections and SNA for DB2 private protocol connections.
- DB2SERV is connected to this DB2 using only SNA.

```
DSNL200I - DISPLAY LOCATION REPORT FOLLOWS -
LOCATION         PRDID    LINKNAME        REQUESTERS SERVERS CONVS
DB2A             DSN05010 LUDB2A                   3       4      9
DB2A             DSN05010 124.38.54.16             2       1      3
DB2SERV          DSN04010 LULA                     1       1      3
DISPLAY LOCATION REPORT COMPLETE
```

The DISPLAY LOCATION command displays information for each remote location that currently is, or once was, in contact with DB2. If a location is displayed with zero conversations, this indicates one of the following:

- Sessions currently exist with the partner location but there are currently no active conversations allocated to any of the sessions.
- Sessions no longer exist with the partner because contact with the partner has been lost.

If you use the DETAIL parameter, each line is followed by information about conversations owned by DB2 system threads, including those used for resynchronization of indoubt units of work.

## The command DISPLAY THREAD

***Displaying information by location:*** Use the LOCATION keyword, followed by a list of location names, to display thread information for particular locations.

You can use an asterisk (*) after the THD and LOCATION keywords just as in the DISPLAY LOCATION command previously described. For example, enter:

```
-DISPLAY THREAD(*) LOCATION(*) DETAIL
```

DB2 returns messages like these:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
   NAME     ST 1 A 2    REQ ID         AUTHID   PLAN     ASID TOKEN
   SERVER   RA  *      2923 DB2BP     ADMF001  DISTSERV 0036    20 3
    V437-WORKSTATION=ARRAKIS, USERID=ADMF001,
          APPLICATION NAME=DB2BP
    V436-PGM=NULLID.SQLC27A4, SEC=201, STMNT=210
    V445-09707265.01BE.889C28200037=20 3 ACCESSING DATA FOR 9.112.12.101
    V447-LOCATION            SESSID       A ST TIME
    V448-9.112.12.101 4      446:1300 5   W S2 9802812045091
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

**Key:**

1 The ST (status) column contains characters that indicate the connection status of the local site. The *TR* indicates that an allied, distributed thread has been established. The *RA* indicates that a distributed thread has been established and is in receive mode. The *RD* indicates that a distributed thread is performing a remote access on behalf of another location (R) and is performing an operation involving DCE services (D). Currently, DB2 supports the optional use of DCE services to authenticate remote users.

2 The A (active) column contains an asterisk indicating that the thread is active within DB2. It is blank when the thread is inactive within DB2 (active or waiting within the application).

3 This LUWID is unique across all connected systems. This thread has a token of *20* (it appears in two places in the display output).

4 This is the location of the data that the local application is accessing. If the RDBNAME is not known, the location column contains either a VTAM LUNAME or a dotted decimal IP address.

5 If the connection uses TCP/IP, the sessid column contains ″local:remote″, where ″local″ specifies DB2's TCP/IP port number and ″remote″ specifies the partner's TCP/IP port number.

For distributed server threads using DRDA access, the NAME column contains SERVER, and the PLAN column contains DISTSERV for all requesters that are not DB2 for MVS Version 3 or later.

For more information about this sample output and connection status codes, see message DSNV404I, DSNV444I, and DSNV446I, in Part 2 of *DB2 Messages and Codes*.

***Displaying information for non-DB2 locations:*** Because DB2 does not receive a location name from non-DB2 locations, you must enter the LUNAME or IP address of the location for which you want to display information. The LUNAME is enclosed by the less-than (<) and greater-than (>) symbols. The IP address is in the dotted decimal format. For example, if you wanted to display information about a non-DB2 DBMS with the LUNAME of LUSFOS2, you would enter the following command:

```
-DISPLAY THREAD (*) LOCATION (<LUSFOS2>)
```

DB2 uses the <LUNAME> notation or dotted decimal format in messages displaying information about non-DB2 requesters.

***Displaying conversation-level information on threads:*** Use the DETAIL keyword with the LOCATION keyword to give you information about conversation activity when distribution information is displayed for active threads. This keyword has no effect on the display of indoubt threads. See Chapter 2 of *DB2 Command Reference* for more information on the DETAIL keyword.

For example, issue:

```
-DISPLAY THREAD(*) LOCATION(*) DETAIL
```

DB2 returns the following message, indicating that the local site application is waiting for a conversation to be allocated in DB2, and a DB2 server that is accessed by a DRDA client using TCP/IP.

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
   NAME     ST A   REQ ID           AUTHID   PLAN     ASID   TOKEN
   TSO      TR *     3 SYSADM       SYSADM   DSNESPRR 002E       2
    V436-PGM=DSNESPRR.DSNESM68, SEC=1, STMNT=116
    V444-DB2NET.LUND0.A238216C2FAE=2 ACCESSING DATA AT
    V446-USIBMSTODB22:LUND1
    V447--LOCATION        SESSID         1 A ST    TIME
    V448--USIBMSTODB22    0000000000000000 V A1 2  9015816504776
   TSO      RA *    11 SYSADM       SYSADM   DSNESPRR 001A      15
    V445-STLDRIV.SSLU.A23555366A29=15 ACCESSING DATA FOR 123.34.101.98
    V447--LOCATION        SESSID           A ST    TIME
    V448--123.34.101.98   446:3171  3        S2    9015611253108
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

**Key:**

**1** The information on this line is part of message DSNV447I. The conversation A (active) column for the server is useful in determining when a DB2 thread is hung and whether processing is waiting in VTAM or in DB2. A value of *W* indicates that the thread is suspended in DB2 and is waiting for notification from VTAM that the event has completed. A value of*V* indicates that control of the conversation is in VTAM.

**2** The information on this line is part of message DSNV448I. The *A* in the conversation ST (status) column for a serving site indicates a conversation is being allocated in DB2. The *1* indicates that the thread uses DB2 private protocol access. A *2* would indicate DRDA access. An *R* in the status column would indicate that the conversation is receiving or waiting to receive a request or reply. An *S* in this column for a server indicates that the application is sending or preparing to send a request or reply.

**3** The information on this line is part of message DSNV448I. The SESSID column has changed as follows. If the connection uses VTAM, the SESSID column contains a VTAM session identifier. If the connection uses TCP/IP, the sessid column contains ″local:remote″, where ″local″ specifies DB2's TCP/IP port number, and ″remote″ specifies the partner's TCP/IP port number.

For more DISPLAY THREAD message information, see messages DSNV447I and DSNV448I, Part 2 of *DB2 Messages and Codes*.

***Monitoring all DBMSs in a transaction:*** The DETAIL keyword of the command DISPLAY THREAD allows you to monitor all of the requesting and serving DBMSs involved in a transaction.

For example, you could monitor an application running at USIBMSTODB21 requesting information from USIBMSTODB22, which must establish conversations with secondary servers USIBMSTODB23 and USIBMSTODB24 to provide the requested information. See Figure 32. In the example, USIBMSTODB21 is considered to be *upstream* from USIBMSTODB22. Similarly, USIBMSTODB22 is considered to be upstream from USIBMSTODB23. Conversely, USIBMSTODB23 and USIBMSTODB22 are *downstream* from USIBMSTODB22 and USIBMSTODB21 respectively.

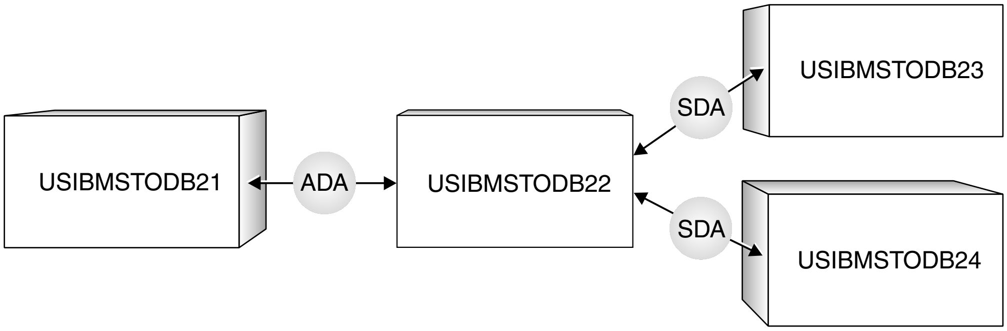


*Figure 32. Example of a DB2 transaction involving four sites. ADA refers to DRDA access, SDA to DB2 private protocol access*

The application running at USIBMSTODB21 is connected to a server at USIBMSTODB22, using DRDA access. If you enter the DISPLAY THREAD command with the DETAIL keyword from USIBMSTODB21, you receive output similar to the following:

```
-DIS THD(*) LOC(*) DET
   DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
   DSNV402I - ACTIVE THREADS -
   NAME     ST A   REQ ID           AUTHID PLAN     ASID   TOKEN
   BATCH    TR *     6 BKH2C        SYSADM YW1019C  0009       2
    V436-PGM=BKH2C.BKH2C, SEC=1, STMNT=4
    V444-USIBMSY.SSLU.A23555366A29=2 ACCESSING DATA AT
    V446-USIBMSTODB22:SSURLU
    V447--LOCATION         SESSID           A ST TIME
    V448--USIBMSTODB22     0000000300000004 V R2 9015611253116
   DISPLAY ACTIVE REPORT COMPLETE
  11:26:23 DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

This output indicates that the application is waiting for data to be returned by the server at USIBMSTODB22.

The server at USIBMSTODB22 is running a package on behalf of the application at USIBMSTODB21, in order to access data at USIBMSTODB23 and USIBMSTODB24 by DB2 private protocol access. If you enter the DISPLAY THREAD command with the DETAIL keyword from USIBMSTODB22, you receive output similar to the following:

```
-DIS THD(*) LOC(*) DET
   DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
   DSNV402I - ACTIVE THREADS -
   NAME     ST A   REQ ID           AUTHID PLAN     ASID   TOKEN
   BATCH    RA *     0 BKH2C        SYSADM YW1019C  0008       2
    V436-PGM=BKH2C.BKH2C, SEC=1, STMNT=4
    V445-STLDRIV.SSLU.A23555366A29=2 ACCESSING DATA FOR
         USIBMSTODB21:SSLU
    V444-STLDRIV.SSLU.A23555366A29=2 ACCESSING DATA AT
    V446-USIBMSTODB23:OSSLU USIBMSTODB24:OSSURLU
    V447--LOCATION         SESSID           A ST TIME
    V448--USIBMSTODB21     0000000300000004   S2 9015611253108
    V448--USIBMSTODB23     0000000600000002   S1 9015611253077
    V448--USIBMSTODB24     0000000900000005 V R1 9015611253907
   DISPLAY ACTIVE REPORT COMPLETE
  11:26:34 DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

This output indicates that the server at USIBMSTODB22 is waiting for data to be returned by the secondary server at USIBMSTODB24.

The secondary server at USIBMSTODB23 is accessing data for the primary server at USIBMSTODB22. If you enter the DISPLAY THREAD command with the DETAIL keyword from USIBMSTODB23, you receive output similar to the following:

```
-DIS THD(*) LOC(*) DET
   DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
   DSNV402I - ACTIVE THREADS -
   NAME     ST A   REQ ID           AUTHID PLAN     ASID   TOKEN
   BATCH    RA *     2 BKH2C        SYSADM YW1019C  0006       1
    V445-STLDRIV.SSLU.A23555366A29=1 ACCESSING DATA FOR
         USIBMSTODB22:SSURLU
    V447--LOCATION         SESSID           A ST TIME
    V448--USIBMSTODB22     0000000600000002 W R1 9015611252369
   DISPLAY ACTIVE REPORT COMPLETE
  11:27:25 DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

This output indicates that the secondary server at USIBMSTODB23 is not currently active.

The secondary server at USIBMSTODB24 is also accessing data for the primary server at USIBMSTODB22. If you enter the DISPLAY THREAD command with the DETAIL keyword from USIBMSTODB24, you receive output similar to the following:

```
-DIS THD(*) LOC(*) DET
   DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
   DSNV402I - ACTIVE THREADS -
   NAME     ST A   REQ ID           AUTHID PLAN     ASID   TOKEN
   BATCH    RA *     2 BKH2C        SYSADM YW1019C  0006       1
    V436-PGM=*.BKH2C, SEC=1, STMNT=1
    V445-STLDRIV.SSLU.A23555366A29=1 ACCESSING DATA FOR
         USIBMSTODB22:SSURLU
    V447--LOCATION         SESSID           A ST TIME
    V448--USIBMSTODB22     0000000900000005   S1 9015611253075
   DISPLAY ACTIVE REPORT COMPLETE
  11:27:32 DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

This output indicates that the secondary server at USIBMSTODB24 is currently active.

It is possible that the conversation status might not change for a long time. The conversation could be hung, or the processing could just be taking a long time. To see whether the conversation is hung, issue DISPLAY THREAD again and compare the new timestamp to the timestamps from previous output messages. If the timestamp is changing, but the status is not, the job is still processing. If it becomes necessary to terminate a distributed job, perhaps because it is hung and has been holding database locks for a long period of time, you can use the CANCEL DDF THREAD command if the thread is in DB2 (whether active or suspended) or the VARY NET TERM command if the thread is within VTAM. See “The command CANCEL THREAD”.

***Displaying threads by LUWIDs:*** Use the LUWID optional keyword, which is only valid when DDF has been started, to display threads by logical unit of work identifiers. The LUWIDs are assigned to the thread by the site that originated the thread.

You can use an asterisk (*) in an LUWID as in a LOCATION name. For example, use -DISPLAY THREAD TYPE(INDOUBT) LUWID(NET1.*) to display all the indoubt threads whose LUWID has a network name of NET1. The command DISPLAY THREAD TYPE(INDOUBT) LUWID(IBM.NEW*) displays all indoubt threads whose LUWID has a network name of ″IBM″ and whose LUNAME begins with ″NEW.″

The DETAIL keyword can also be used with the DISPLAY THREAD LUWID command to show the status of every conversation connected to each thread displayed and to indicate whether a conversation is using DRDA access or DB2 private protocol access.

To issue this command enter:

```
-DIS THD(*) LUWID (luwid) DETAIL
```

DB2 returns the following message and output similar to the sample output provided:

```
-DIS THD(*) LUWID (luwid) DET
  DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
  DSNV402I - ACTIVE THREADS -
  NAME     ST A   REQ ID          AUTHID   PLAN     ASID   TOKEN
  BATCH    TR       5 TC3923S0     SYSADM   TC392    000D       2
   V436-PGM=*.TC3923S0, SEC=1, STMNT=116
   V444-DB2NET.LUNSITE0.A11A7D7B2057=2  1 ACCESSING DATA AT
   V446-USIBMSTODB22:LUNSITE1
   V447--LOCATION         SESSID           A ST    TIME
   V448--USIBMSTODB22     00C3F4228C5A244C   S2 2  8929612225354
  DISPLAY ACTIVE REPORT COMPLETE
  DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

**Key:**

**1** In the display output above, you can see that the LUWID has been assigned a token of *2*. You can use this token instead of the long version of the LUWID to cancel or display the given thread. For example:

```
-DIS THD(*) LUWID(2) DET
```

**2** In addition, the status column for the serving site contains a value of *S2*. The *S* means that this thread can send a request or response, and the *2* means that this is an DRDA access conversation.

## The command CANCEL THREAD

You can use the command CANCEL THREAD to terminate threads that are active or suspended in DB2. The command has no effect if the thread is not active or suspended in DB2. If the thread is suspended in VTAM, you can use VTAM commands to terminate the conversations, as described in “Using VTAM commands to cancel threads” on page 319.

A database access thread can also be in the prepared state waiting for the commit decision from the coordinator. When you issue CANCEL THREAD for a database access thread in the prepared state, the thread is converted from active to indoubt. The conversation with the coordinator, and all conversations with downstream participants, are terminated and message DSNL450I is returned. The resources held by the thread are not released until the indoubt state is resolved. This is accomplished automatically by the coordinator or by using the command RECOVER INDOUBT. See “Resolving indoubt units of recovery” on page 363 for more information.

DISPLAY THREAD can be used to determine if a thread is hung in DB2 or VTAM. If in VTAM, there is no reason to use the CANCEL command.

Using CANCEL THREAD requires SYSOPR authority or higher.

When the command is entered at the DB2 system that has a database access thread servicing requests from a DB2 system that owns the allied thread, the database access thread is terminated. Any active SQL request, and all later requests, from the allied thread result in a ″resource not available″ return code.

To issue this command enter:

```
-CANCEL THREAD (token)
```

Or, if you like, you can use the following version of the command with either the token or LUW ID:

```
-CANCEL DDF THREAD (token or luwid)
```

The *token* is a 1- to 5-character number that identifies the thread. When DB2 schedules the thread for termination, you will see one of the following messages:

```
DSNL010I - DDF THREAD token or luwid HAS BEEN CANCELED
```

for a distributed thread, or

```
DSNV426I - csect THREAD token HAS BEEN CANCELED
```

for a non-distributed thread.

For more information about CANCEL THREAD, see Chapter 2 of *DB2 Command Reference*.

***Diagnostic dumps:*** CANCEL THREAD allows you to specify that a diagnostic dump be taken.

For more detailed information about diagnosing DDF failures see Part 3 of *DB2 Diagnosis Guide and Reference*.

***Messages:*** As a result of entering CANCEL THREAD, the following messages can be displayed:

DSNL009I
DSNL010I
DSNL022I

## Using VTAM commands to cancel threads

If the command CANCEL THREAD does not terminate the thread, it is possible that it is hung up in VTAM, not in DB2. Use the VTAM VARY NET,TERM command to cancel the thread's VTAM sessions. The VTAM commands only work with SNA VTAM connections, not TCP/IP connections.

To do this, you need to know the VTAM session IDs that correspond to the thread. Follow these steps:

1. Issue the DB2 command DISPLAY THREAD(*nnnn*) LOC(*) DETAIL.

   This gives you the VTAM session IDs that must be canceled. As is shown in the DISPLAY THREAD output in Figure 33, these sessions are identified by the column header SESSID.

```
-DIS THD LOC(*) DETAIL

DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID    PLAN      ASID    TOKEN
BATCH    TR *     5 BKH2C        SYSADM    BKH2      000D      123
V436-PGM=*.BKH2C, SEC, STMNT=116
 V445-DB2NET.LUND0.9F6D9F459E92=123 ACCESSING DATA FOR
      USIBMSTODB21:LUND1
 V447--LOCATION         SESSID             A ST TIME
 V448--USIBMSTODB21   v 00D3590EA1E89701     S1 8832108460302
 V448--USIBMSTODB21     00D3590EA1E89822   V R1 8832108460431
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

*Figure 33. Sample DISPLAY THREAD output*

2. Record positions 3 through 16 of SESSID for the threads to be canceled. (In the DISPLAY THREAD output above, the values are D3590EA1E89701 and D3590EA1E89822.)
3. Issue the VTAM command DISPLAY NET to display the VTAM session IDs (SIDs). The ones you want to cancel match the SESSIDs in positions 3 through 16. In figure Figure 34, the corresponding session IDs are shown in bold.

```
D NET,ID=LUND0,SCOPE=ACT

IST097I DISPLAY ACCEPTED
IST075I NAME = LUND0, TYPE = APPL
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
  IST171I ACTIVE SESSIONS = 0000000010, SESSION REQUESTS = 0000
  IST206I SESSIONS:
  IST634I NAME     STATUS           SID          SEND   RECV
  IST635I LUND1    ACTIV-S     D24B171032B76E65  0051   0043
  IST635I LUND1    ACTIV-S     D24B171032B32545  0051   0043
  IST635I LUND1    ACTIV-S     D24B171032144565  0051   0043
  IST635I LUND1    ACTIV-S     D24B171032B73465  0051   0043
  IST635I LUND1    ACTIV-S     D24B171032B88865  0051   0043
  IST635I LUND1    ACTIV-R     D2D3590EA1E89701  0022   0031
  IST635I LUND1    ACTIV-R     D2D3590EA1E89802  0022   0031
  IST635I LUND1    ACTIV-R     D2D3590EA1E89809  0022   0031
  IST635I LUND1    ACTIV-R     D2D3590EA1E89821  0022   0031
  IST635I LUND1    ACTIV-R     D2D3590EA1E89822  0022   0031
  IST314I END
```

*Figure 34. Sample output for VTAM DISPLAY NET command*

4. Issue the VTAM command VARY NET,TERM SID= for each of the VTAM SIDs associated with the DB2 thread. For more information about VTAM commands, see *VTAM for MVS/ESA Operation*.

## Monitoring and controlling stored procedures

Stored procedures are user-written SQL programs that run at a DB2 server. Stored procedures can run in DB2-established or WLM-established address spaces. To monitor and control stored procedures in WLM-established address spaces, you might need to use WLM commands, rather than DB2 commands. When you execute a WLM command on an MVS system that is part of a Sysplex, the scope of that command is the Sysplex.

This section discusses the following topics:
- "Displaying information about stored procedures and their environment"
- "Refreshing the environment for stored procedures or user-defined functions" on page 322
- "Obtaining diagnostic information about stored procedures" on page 323

For more information about stored procedures, see Part 6 of *DB2 Application Programming and SQL Guide*.

### Displaying information about stored procedures and their environment

Use the DB2 commands DISPLAY PROCEDURE and DISPLAY THREAD to obtain information about a stored procedure while it is running. In the WLM-established environment, use the MVS command DISPLAY WLM to obtain information about the application environment in which a stored procedure runs.

***The DB2 command DISPLAY PROCEDURE:*** This command can display the following information about stored procedures:

- Status (started, stop-queue, stop-reject, stop-abend)
- Number of requests currently running and queued
- Maximum number of threads running a stored procedure load module and queued
- Count of timed-out SQL CALLs

The following command displays information about all stored procedures in all schemas that have been accessed by DB2 applications:

```
-DISPLAY PROCEDURE
```

```
DSNX940I csect - DISPLAY PROCEDURE REPORT FOLLOWS-

------ SCHEMA=PAYROLL
PROCEDURE          STATUS    ACTIVE  QUEUED  MAXQUE  TIMEOUT  WLM_ENV
PAYRPRC1           STARTED        0       0       1        0  PAYROLL
PAYRPRC2           STOPQUE        0       5       5        3  PAYROLL
PAYRPRC3           STARTED        2       0       6        0  PAYROLL
USERPRC4           STOPREJ        0       0       1        0  SANDBOX

------ SCHEMA=HRPROD
PROCEDURE          STATUS    ACTIVE  QUEUED  MAXQUE  TIMEOUT  WLM_ENV
HRPRC1             STARTED        0       0       1        0  HRPROCS
HRPRC2             STOPREJ        0       0       1        0  HRPROCS
DISPLAY PROCEDURE REPORT COMPLETE
```

In this example there are two schemas (PAYROLL and HRPROD) that have been accessed by DB2 applications. You can also display information about specific stored procedures.

***The DB2 command DISPLAY THREAD:*** This command tells whether:
- A thread is waiting for a stored procedure to be scheduled
- A thread is executing within a stored procedure

Here is an example of DISPLAY THREAD output that shows a thread that is executing a stored procedure:

```
!display thread(*) det
DSNV401I ! DISPLAY THREAD REPORT FOLLOWS -
DSNV402I ! ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
BATCH    SP       3 CALLWLM      SYSADM   PLNAPPLX 0022     5
  V436-PGM=*.MYPROG, SEC=2, STMNT=1
  V429 CALLING PROCEDURE=SYSADM  .WLMSP              ,
       PROC=V61AWLM1, ASID=0085, WLM_ENV=WLMENV1
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I ! DSNVDT '-DIS THD' NORMAL COMPLETION
```

The SP status indicates that the thread is executing within the stored procedure. An SW status indicates that the thread is waiting for the stored procedure to be scheduled.

Here is an example of DISPLAY THREAD output that shows a thread that is executing a user-defined function:

```
!display thd(*) det
DSNV401I ! DISPLAY THREAD REPORT FOLLOWS -
DSNV402I ! ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
BATCH    SP      27 LI33FN1      SYSADM   DSNTEP3  0021     4
 V436-PGM=*.MYPROG, SEC=2, STMNT=1
 V429 CALLING FUNCTION =SYSADM  .FUNC1             ,
      PROC=V61AWLM1, ASID=0085, WLM_ENV=WLMENV1
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I ! DSNVDT '-DISPLAY THD' NORMAL COMPLETION
```

***The MVS command DISPLAY WLM:*** Use the command DISPLAY WLM to determine the status of an application environment in which a stored procedure runs. The output from DISPLAY WLM lets you determine whether a stored procedure can be scheduled in an application environment.

For example, you can issue this command to determine the status of application environment WLMENV1:

```
D WLM,APPLENV=WLMENV1
```

You might get results like this:

```
IWM029I  15.22.22  WLM DISPLAY
APPLICATION ENVIRONMENT NAME     STATE     STATE DATA
WLMENV1                          AVAILABLE
ATTRIBUTES: PROC=DSNWLM1 SUBSYSTEM TYPE: DB2
```

The output tells you that WLMENV1 is available, so WLM can schedule stored procedures for execution in that environment.

## Refreshing the environment for stored procedures or user-defined functions

Depending on what has changed in a stored procedures environment, you might need to perform one or more of these tasks:

- Refresh Language Environment®.

  Do this when someone has modified a load module for a stored procedure, and that load module is cached in a stored procedures address space. When you refresh Language Environment, the cached load module is purged. On the next invocation of the stored procedure, the new load module is loaded.
- Restart a stored procedures address space.

  You might stop and then start a stored procedures address space because you need to make a change to the startup JCL for a stored procedures address space. You might need to start a stored procedures address space because the address space has abnormally terminated.

The method that you use to perform these tasks for stored procedures depends on whether you are using WLM-established or DB2-established address spaces.

***For DB2-established address spaces:*** Use the DB2 commands -START PROCEDURE and -STOP PROCEDURE to perform all of these tasks.

***For WLM-established address spaces:*** If WLM is operating in goal mode:

- Use the MVS command

  ```
  VARY WLM,APPLENV=name,REFRESH
  ```

  to refresh Language Environment when you need to load a new version of a stored procedure. *name* is the name of a WLM application environment associated with a group of stored procedures. This means that when you execute this command, you affect all stored procedures associated with the application environment.
- Use the MVS command

  ```
  VARY WLM,APPLENV=name,QUIESCE
  ```

  to stop all stored procedures address spaces associated with WLM application environment *name*.
- Use the MVS command

  ```
  VARY WLM,APPLENV=name,RESUME
  ```

  to start all stored procedures address spaces associated with WLM application environment *name*.

  You also need to use the VARY WLM command with the RESUME option when WLM puts an application environment in the unavailable state. An application environment in which stored procedures run becomes unavailable when WLM detects 5 abnormal terminations within 10 minutes. When an application environment is in the unavailable state, WLM does not schedule stored procedures for execution in it.

  See *OS/390 MVS Planning: Workload Management* for more information on the command VARY WLM.

If WLM is operating in compatibility mode:

- Use the MVS command

  ```
  CANCEL address-space-name
  ```

to stop a WLM-established stored procedures address space.

- Use the MVS command

  ```
  START address-space-name
  ```

  to start a WLM-established stored procedures address space.

  In compatibility mode, you must stop and start stored procedures address spaces when you need to refresh Language Environment.

### Obtaining diagnostic information about stored procedures

If the startup procedures for your stored procedures address spaces contain a DD statement for CEEDUMP, Language Environment writes a small diagnostic dump to CEEDUMP when a stored procedure terminates abnormally. The output waits to print until the stored procedures address space terminates.

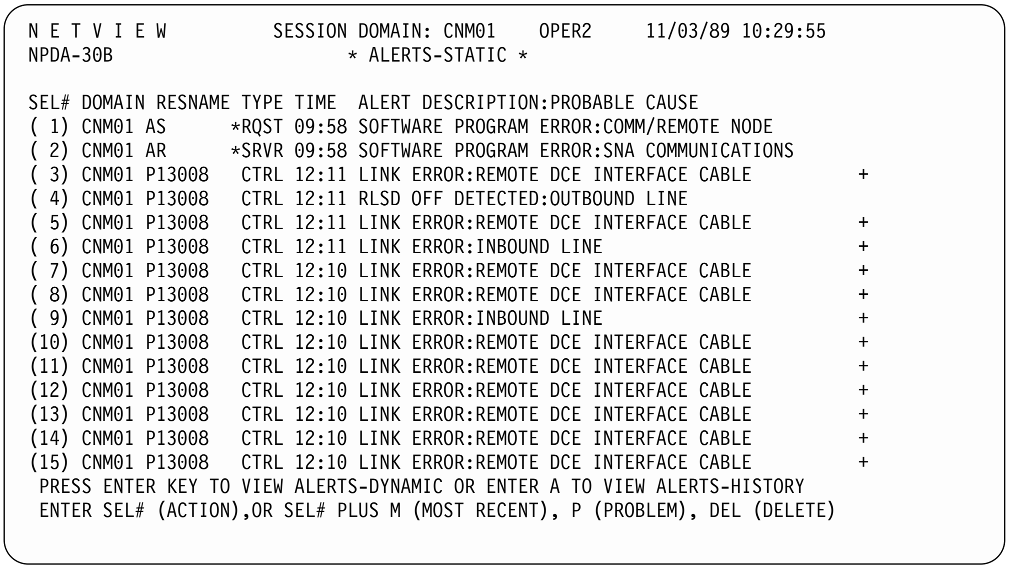
You can obtain the dump information by stopping the stored procedures address space in which the stored procedure is running. See "Refreshing the environment for stored procedures or user-defined functions" on page 322 for information on how to stop and start stored procedures address spaces in the DB2-established and WLM-established environments.

## Using NetView® to monitor errors in the network

The NetView program lets you have a single focal point from which to view problems in the network. DDF sends an alert to NetView when a remote location is either involved in the cause of the failure or affected by the failure. The following major events generate alerts:

- Conversation failures
- Distributed security failures
- DDF abends
- DDM protocol errors
- Database access thread abends
- Distributed allied thread abends

Alerts for DDF are displayed on NetView's Hardware Monitor panels and are logged in the hardware monitor database. Figure 35 on page 324 is an example of the Alerts-Static panel in NetView.

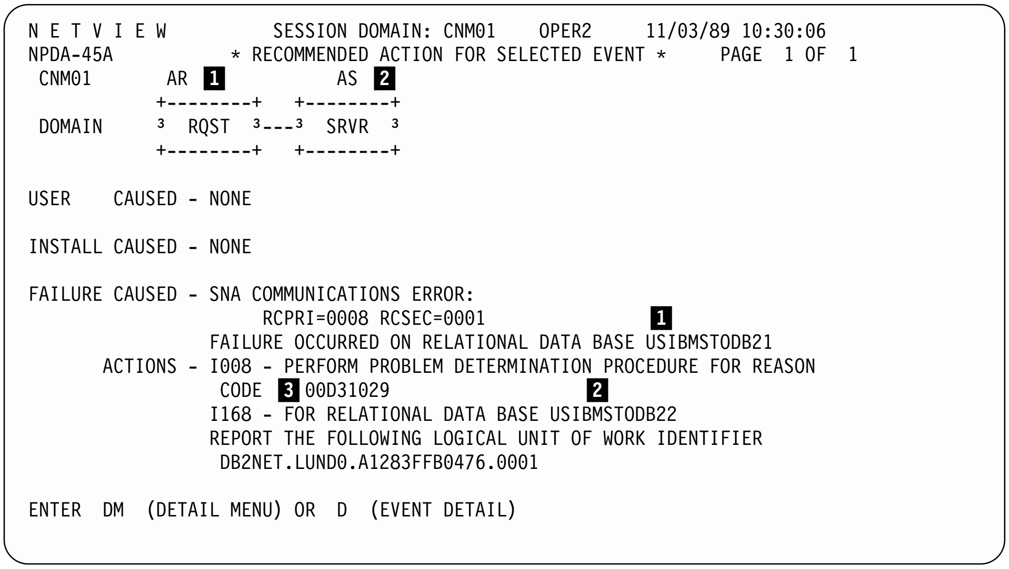


```
N E T V I E W         SESSION DOMAIN: CNM01    OPER2     11/03/89 10:29:55
NPDA-30B                       * ALERTS-STATIC *

SEL# DOMAIN RESNAME TYPE TIME  ALERT DESCRIPTION:PROBABLE CAUSE
( 1) CNM01 AS      *RQST 09:58 SOFTWARE PROGRAM ERROR:COMM/REMOTE NODE
( 2) CNM01 AR      *SRVR 09:58 SOFTWARE PROGRAM ERROR:SNA COMMUNICATIONS
( 3) CNM01 P13008   CTRL 12:11 LINK ERROR:REMOTE DCE INTERFACE CABLE          +
( 4) CNM01 P13008   CTRL 12:11 RLSD OFF DETECTED:OUTBOUND LINE
( 5) CNM01 P13008   CTRL 12:11 LINK ERROR:REMOTE DCE INTERFACE CABLE          +
( 6) CNM01 P13008   CTRL 12:11 LINK ERROR:INBOUND LINE                        +
( 7) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE          +
( 8) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE          +
( 9) CNM01 P13008   CTRL 12:10 LINK ERROR:INBOUND LINE                        +
(10) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE          +
(11) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE          +
(12) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE          +
(13) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE          +
(14) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE          +
(15) CNM01 P13008   CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE          +
 PRESS ENTER KEY TO VIEW ALERTS-DYNAMIC OR ENTER A TO VIEW ALERTS-HISTORY
 ENTER SEL# (ACTION),OR SEL# PLUS M (MOST RECENT), P (PROBLEM), DEL (DELETE)
```

*Figure 35. Alerts-static panel in NetView. DDF errors are denoted by the resource name AS (server) and AR (requester). For DB2-only connections, the resource names would be RS (server) and RQ (requester).*

To see the recommended action for solving a particular problem, enter the selection number, then press ENTER. This displays the Recommended Action for Selected Event panel shown in Figure 36.



```
N E T V I E W         SESSION DOMAIN: CNM01    OPER2     11/03/89 10:30:06
NPDA-45A           * RECOMMENDED ACTION FOR SELECTED EVENT *     PAGE  1 OF  1
 CNM01       AR 1          AS 2
           +--------+   +--------+
 DOMAIN    ³  RQST  ³---³  SRVR  ³
           +--------+   +--------+

USER    CAUSED - NONE

INSTALL CAUSED - NONE

FAILURE CAUSED - SNA COMMUNICATIONS ERROR:
                      RCPRI=0008 RCSEC=0001              1
                 FAILURE OCCURRED ON RELATIONAL DATA BASE USIBMSTODB21
       ACTIONS - I008 - PERFORM PROBLEM DETERMINATION PROCEDURE FOR REASON
                  CODE 3 00D31029                  2
                 I168 - FOR RELATIONAL DATA BASE USIBMSTODB22
                 REPORT THE FOLLOWING LOGICAL UNIT OF WORK IDENTIFIER
                  DB2NET.LUND0.A1283FFB0476.0001

ENTER  DM  (DETAIL MENU) OR  D  (EVENT DETAIL)
```

*Figure 36. Recommended action for selected event panel in NetView. In this example, the AR (USIBMSTODB21) is reporting the problem, which is affecting the AS (USIBMSTODB22).*

**Key:**

1 The system reporting the error. The system reporting the error is always on the left side of the panel. That system's name appears first in the messages. Depending on who is reporting the error, either the LUNAME or the location name is used.

**2** The system affected by the error. The system affected by the error is always displayed to the right of the system reporting the error. The affected system's name appears second in the messages. Depending on what type of system is reporting the error, either the LUNAME or the location name is used.

If no other system is affected by the error, then this system will not appear on the panel.

**3** DB2 reason code. For information about DB2 reason codes, see Part 3 of *DB2 Messages and Codes*. For diagnostic information, see Part 3 of *DB2 Diagnosis Guide and Reference*.

For more information about using NetView, see *NetView User's Guide*.

## Stopping DDF

**General-use Programming Interface**

You need SYSOPR authority or higher to stop the distributed data facility. Use one of the following commands:

```
-STOP DDF MODE (QUIESCE)
-STOP DDF MODE (FORCE)
```

Use the QUIESCE option whenever possible; it is the default. With QUIESCE, the STOP DDF command does not complete until all VTAM or TCP/IP requests have completed. In this case, no resynchronization work is necessary when you restart DDF. If there are indoubt units of work that require resynchronization, the QUIESCE option produces message DSNL035I. Use the FORCE option only when you must stop DDF quickly. Restart times are longer if you use FORCE.

When DDF is stopped with the FORCE option, and DDF has indoubt thread responsibilities with remote partners, one or both of messages DSNL432I and DSNL433I is generated.

DSNL432I shows the number of threads that DDF has coordination responsibility over with remote participants who could have indoubt threads. At these participants, database resources that are unavailable because of the indoubt threads remain unavailable until DDF is started and resolution occurs.

DSNL433I shows the number of threads that are indoubt locally and need resolution from remote coordinators. At the DDF location, database resources are unavailable because the indoubt threads remain unavailable until DDF is started and resolution occurs.

To force the completion of outstanding VTAM or TCP/IP requests, use the FORCE option, which cancels the threads associated with distributed requests.

When the FORCE option is specified with STOP DDF, database access threads in the prepared state that are waiting for the commit or abort decision from the coordinator are logically converted to the indoubt state. The conversation with the coordinator is terminated. If the thread is also a coordinator of downstream participants, these conversations are terminated. Automatic indoubt resolution is initiated when DDF is restarted. See "Resolving indoubt units of recovery" on page 363 for more information on this topic.

The STOP DDF command causes the following messages to appear:

```
DSNL005I - DDF IS STOPPING
DSNL006I - DDF STOP COMPLETE
```

If the distributed data facility has already been stopped, the STOP DDF command fails and message DSNL002I - DDF IS ALREADY STOPPED appears.

***Stopping DDF using VTAM commands:*** Another way to force DDF to stop is to issue the VTAM VARY NET,INACT command. This command makes VTAM unavailable and terminates DDF. VTAM forces the completion of any outstanding VTAM requests immediately.

The syntax for the command is as follows:

```
VARY NET,INACT,ID=db2lu,FORCE
```

where *db2lu* is the VTAM LU name for the local DB2 system.

When DDF has stopped, the following command must be issued before -START DDF can be attempted:

```
VARY NET,ACT,ID=db2lu
```

**End of General-use Programming Interface**

# Controlling traces

These traces can be used for problem determination:

DB2 trace
IMS attachment facility trace
CICS trace
Three TSO attachment facility traces
CAF trace stream
OS/390 RRS trace stream
MVS component trace used for IRLM

## Controlling the DB2 trace

**General-use Programming Interface**

DB2 trace allows you to trace and record subsystem data and events. There are five different types of trace. For classes of events traced by each type see the description of the START TRACE command in Chapter 2 of *DB2 Command Reference*. For more information about the trace output produced, see “Appendix D. Interpreting DB2 trace output” on page 981. In brief, DB2 records the following types of data:

**Statistics**
Data that allows you to conduct DB2 capacity planning and to tune the entire set of DB2 programs.

**Accounting**
Data that allows you to assign DB2 costs to individual authorization IDs and to tune individual programs.

**Performance**
Data about subsystem events, which can be used to do program, resource, user, and subsystem-related tuning.

**Audit** Data that can be used to monitor DB2 security and access to data.

**Monitor**
Data that is available for use by DB2 monitor application programs.

DB2 provides commands for controlling the collection of this data. To use the trace commands you must have one of the following types of authority:

- SYSADM or SYSOPR authority
- Authorization to issue start and stop trace commands (the TRACE privilege)
- Authorization to issue the display trace command (the DISPLAY privilege).

The trace commands include:

**START TRACE**
Invokes one or more different types of trace.

**DISPLAY TRACE**
Displays the trace options that are in effect.

**STOP TRACE**
Stops any trace that was started by either the START TRACE command or the parameters specified when installing or migrating.

**MODIFY TRACE**
Changes the trace events (IFCIDs) being traced for a specified active trace.

Several parameters can be specified to further qualify the scope of a trace. Specific events within a trace type can be traced as well as events within specific DB2 plans, authorization IDs, resource manager IDs and location. The destination to which trace data is sent can also be controlled. For a discussion of trace commands, see Chapter 2 of *DB2 Command Reference*.

When you install DB2, you can request that any trace type and class start automatically when DB2 starts. For information on starting traces automatically, see Part 2 of *DB2 Installation Guide*.

**End of General-use Programming Interface**

## Diagnostic traces for the attachment facilities

The following trace facilities are for diagnostic purposes only:

- IMS provides a trace facility that shows the flow of requests across the connections from the control and dependent regions to DB2. The trace is recorded on the IMS log if the appropriate options are specified, and then it is printed with DFSERA10 plus a formatting exit module. For more information about this trace facility, see *IMS Utilities Reference: System*.

  In addition, the IMS attachment facility of DB2 provides an internal wrap-around trace table that is always active. When certain unusual error conditions occur, these trace entries are externalized on the IMS log.
- You can use the CICS trace facility to trace the CICS attachment facility.

  Use the transaction CETR to control the CICS trace facility. CETR gives you a series of menus that you can use to set CICS trace options. For CICS 4.1 and later, to trace the CICS attachment facility, set these values in the Component Trace Options panel:
  – For CICS 4.1, specify the value 2 in the FC field.
  – For later releases, specify the value 2 in the RI field.

For information about using the CETR transaction to control CICS tracing, see *CICS for MVS/ESA CICS-Supplied Transactions*.

- The TSO attachment facility provides three tracing mechanisms:
  The DSN trace stream
  The CLIST trace facility
  The SPUFI trace stream
- The call attachment facility trace stream uses the same *ddname* as the TSO DSN trace stream, but is independent of TSO.
- The RRSAF trace stream uses the same *ddname* as the TSO DSN trace stream, but is independent of TSO. An RRSAF internal trace will be included in any ABEND dump produced by RRSAF. This tracing facility provides a history of RRSAF usage that can aid in diagnosing errors in RRSAF.

## Diagnostic trace for the IRLM

The following MVS commands control diagnostic traces for the IRLM:

**MODIFY** *irlmproc***,SET,TRACE**
Sets dynamically the maximum number of trace buffers for each trace type. This value is used only when the external component trace writer is not activated.

**MODIFY** *irlmproc***,STATUS,TRACE**
Displays the status of traces and the number of trace buffers used for each trace type. Also displays whether or not the external component trace writer is active for the trace.

**START** *irlmproc***,TRACE=YES**
Captures traces in wrap-around IRLM buffers at IRLM startup.

**TRACE CT**
Starts, stops, or modifies a diagnostic trace for IRLM. The TRACE CT command does not know about traces that are started automatically during IRLM startup.

Recommendations:

- Do not use the external component trace writer to write traces to the data set.
- Activate all traces during IRLM startup. Use the command START *irlmproc*,TRACE=YES to activate all traces.

See Chapter 2 of *DB2 Command Reference* for detailed information.

## Controlling the resource limit facility (governor)

**General-use Programming Interface**

The governor allows the system administrator to limit the amount of time permitted for the execution of the SELECT, UPDATE, DELETE, and INSERT dynamic SQL statements.

DB2 provides these commands for controlling the governor:

**START RLIMIT**
Starts the governor and identifies a resource limit specification table. You can also use START RLIMIT to switch resource limit specification tables.

**DISPLAY RLIMIT**
: Displays the current status of the governor. If the governor has been started, it also identifies the resource limit specification table.

**STOP RLIMIT**
: Stops the governor and removes any set limits.

The limits are defined in resource limit specification tables and can vary for different users. One resource limit specification table is used for each invocation of the governor and is identified on the START RLIMIT command.

See “Resource limit facility (governor)” on page 581 for more information about the governor.

**End of General-use Programming Interface**

When you install DB2, you can request that the governor start automatically when DB2 starts. For information on starting the governor automatically, see Part 2 of *DB2 Installation Guide*.

## Changing subsystem parameter values

You can modify the values of subsystem parameters dynamically even while DB2 is running by following these steps:

1. Run the installation process in UPDATE mode, specifying any new parameter values. This process produces a new DSNTIJUZ job with the new values; it also saves these values in the file specified as the output member name on panel DSNTIPA1.
2. Assemble and link-edit the DSNTIJUZ job produced in Step 1, and then submit the job to create the new load module with the new subsystem parameter values.
3. Issue the SET SYSPARM command to change the subsystem parameters dynamically:

   ```
   SET SYSPARM LOAD(load-module-name)
   ```

   where you specify the *load-module-name* to be the same as the output member name in Step 1.

   If you specify the load module name that was used during installation, you can issue this command:

   ```
   SET SYSPARM RELOAD
   ```

   For further information, see Part 2 of *DB2 Installation Guide* and Chapter 2 of *DB2 Command Reference*.

# Chapter 18. Managing the log and the bootstrap data set

The DB2 log registers data changes and significant events as they occur. The bootstrap data set (BSDS) is a repository of information about the data sets that contain the log.

DB2 writes each log record to a disk data set called the *active log*. When the active log is full, DB2 copies its contents to a disk or tape data set called the *archive log*. That process is called *offloading*. This chapter describes:

“How database changes are made”
“Establishing the logging environment” on page 333
“Managing the bootstrap data set (BSDS)” on page 341
“Discarding archive log records” on page 343

For information about the physical and logical records that make up the log, see “Appendix C. Reading log records” on page 957. That appendix also contains information about how to write a program to read log records.

## How database changes are made

Before you can fully understand how logging works, you need to be familiar with how database changes are made to ensure consistency. In this section, we discuss *units of recovery* and *rollbacks*.

### Units of recovery

A *unit of recovery* is the work, done by a single DB2 DBMS for an application, that changes DB2 data from one point of consistency to another. A *point of consistency* (also, *sync point* or *commit point*) is a time when all recoverable data that an application program accesses is consistent with other data. (For an explanation of maintaining consistency between DB2 and another subsystem such as IMS or CICS, see “Consistency with other systems” on page 359.)

A unit of recovery begins with the first change to the data after the beginning of the job or following the last point of consistency and ends at a later point of consistency. An example of units of recovery within an application program is shown in Figure 37.

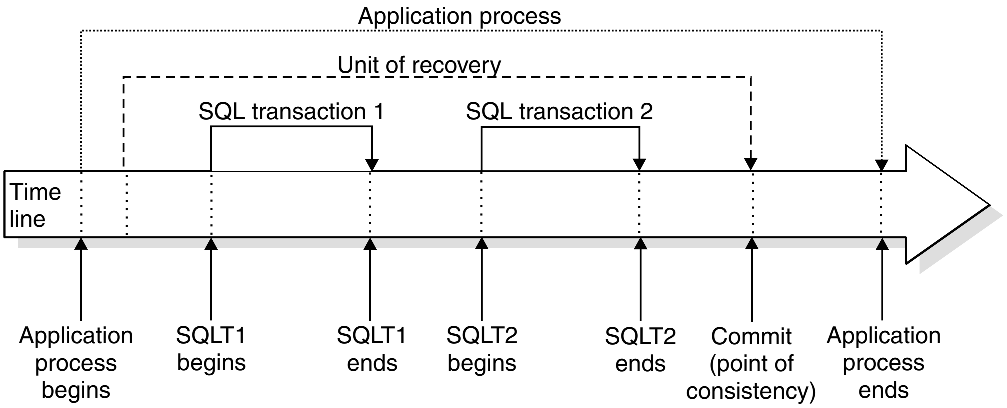


*Figure 37. A unit of recovery within an application process*

In this example, the application process makes changes to databases at SQL transaction 1 and 2. The application process can include a number of units of recovery or just one, but any complete unit of recovery ends with a commit point.

For example, a bank transaction might transfer funds from account A to account B. First, the program subtracts the amount from account A. Next, it adds the amount to account B. After subtracting the amount from account A, the two accounts are inconsistent. These accounts are inconsistent until the amount is added to account B. When both steps are complete, the program can announce a point of consistency and thereby make the changes visible to other application programs.

Normal termination of an application program automatically causes a point of consistency. The SQL COMMIT statement causes a point of consistency during program execution under TSO. A sync point causes a point of consistency in CICS and IMS programs.

## Rolling back work

If failure occurs within a unit of recovery, DB2 backs out any changes to data, returning the data to its state at the start of the unit of recovery; that is, DB2 *undoes* the work. The events are shown in Figure 38. The SQL ROLLBACK statement, and deadlocks and timeouts (reported as SQLCODE -911, SQLSTATE 40001), cause the same events.

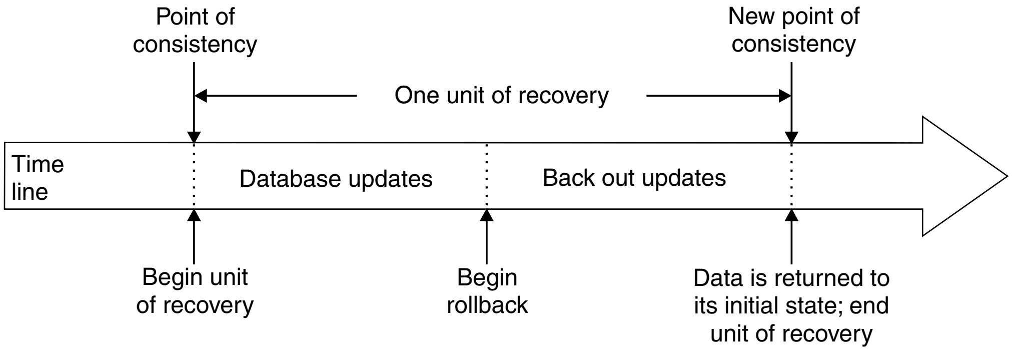


*Figure 38. Unit of recovery (rollback)*

The effects of inserts, updates, and deletes to large object (LOB) values are backed out along with all the other changes made during the unit of work being rolled back, even if the LOB values that were changed reside in a LOB table space with the LOG NO attribute.

An operator or an application can issue the CANCEL THREAD command with the NOBACKOUT option to cancel long running threads without backing out data changes. As a result, DB2 does not read the log records and does not write or apply the compensation log records. After CANCEL THREAD NOBACKOUT processing, DB2 marks all objects associated with the thread as refresh pending (REFP) and puts the objects in a logical page list (LPL). For information about how to reset the REFP status, see *DB2 Utility Guide and Reference*.

The NOBACKOUT request might fail for either of the following two reasons:

- DB2 does not completely back out updates of the catalog or directory (message DSNI032I with reason 00C900CC).
- The thread is part of a global transaction (message DSNV439I).

## Establishing the logging environment

The DB2 logging environment is established by using installation panels to specify options, such as whether to have dual active logs (strongly recommended), what media to use for archive log volumes, and how many log buffers to have. For details of the installation process, see Part 2 of *DB2 Installation Guide*.

## Creation of log records

Log records typically go through the following cycle:

1. DB2 registers changes to data and significant events in recovery log records.
2. DB2 processes recovery log records and breaks them into segments if necessary.
3. Log records are placed sequentially in *output log buffers*, which are formatted as VSAM control intervals (CIs). Each log record is identified by a continuously increasing RBA in the range 0 to $2^{48}$-1, where $2^{48}$ represents 2 to the 48th power. (In a data sharing environment, a log record sequence number (LRSN) is used to identify log records. See *DB2 Data Sharing: Planning and Administration* for more information.)
4. The CIs are written to a set of predefined disk *active log data sets*, which are used sequentially and recycled.
5. As each active log data set becomes full, its contents are automatically *offloaded* to a new *archive log data set*.

If you change or create data that is compressed, the data logged is also compressed. Changes to compressed rows like inserts, updates, and deletes are also logged as compressed data.

## Retrieval of log records

Log records are retrieved through the following events:

1. A log record is requested using its RBA.
2. DB2 searches for the log record in the locations listed below, in the order given:
   a. The log buffers.
   b. The active logs. The bootstrap data set registers which log RBAs apply to each active or archive log data set. If the record is in an active log, DB2 dynamically acquires a buffer, reads one or more CIs, and returns one record for each request.
   c. The archive logs. DB2 determines which archive volume contains the CIs, dynamically allocates the archive volume, acquires a buffer, and reads the CIs.

## Writing the active log

The log buffers are written to an active log data set when they become full, when the write threshold is reached (as specified on the DSNTIPL panel), or, more often, when the DB2 subsystem forces the log buffer to be written (such as, at commit time). In the last case, the same control interval can be written several times to the same location. The use of dual active logs increases the reliability of recovery.

When DB2 is initialized, the active log data sets named in the BSDS are dynamically allocated for *exclusive* use by DB2 and remain allocated exclusively to DB2 (the data sets were allocated as DISP=OLD) until DB2 terminates. Those active log data sets cannot be replaced, nor can new ones be added, without terminating and restarting DB2. The size and number of log data sets is indicated by what was specified by installation panel DSNTIPL.

## Writing the archive log (offloading)

The process of copying active logs to archive logs is called *offloading*. The relation of offloading to other logging events is shown schematically in Figure 39.

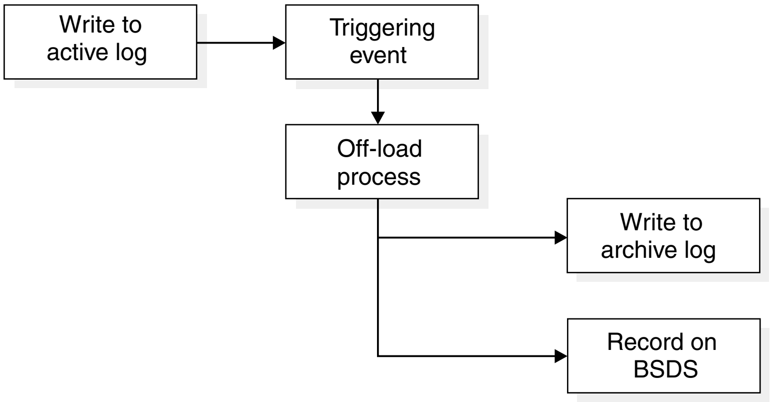


*Figure 39. The offloading process*

### Triggering offload

An offload of an active log to an archive log can be triggered by several events. The most common are when:

- An active log data set is full
- Starting DB2 and an active log data set is full
- The command ARCHIVE LOG is issued

An offload is also triggered by two uncommon events:

- An error occurring while writing to an active log data set. The data set is truncated before the point of failure, and the record that failed to write becomes the first record of the next data set. An offload is triggered for the truncated data set as in normal end-of-file. If there are dual active logs, both copies are truncated so the two copies remain synchronized.
- Filling of the last unarchived active log data set. Message DSNJ110E is issued, stating the percentage of its capacity in use; IFCID trace record 0330 is also issued if statistics class 3 is active. If all active logs become full, DB2 stops processing until offloading occurs and issues this message:

  ```
  DSNJ111E - OUT OF SPACE IN ACTIVE LOG DATA SETS
  ```

### The offloading process

During the process, DB2 determines which data set to offload. Using the last log RBA offloaded, as registered in the BSDS, DB2 calculates the log RBA at which to start. DB2 also determines the log RBA at which to end, from the RBA of the last log record in the data set, and registers that RBA in the BSDS.

When all active logs become full, the DB2 subsystem runs an offload and halts processing until the offload is completed. If the offload processing fails when the active logs are full, then DB2 cannot continue doing any work that requires writing to the log. For additional information, see “Active log failure” on page 423.

When an active log is ready to be offloaded, a request can be sent to the MVS console operator to mount a tape or prepare a disk unit. The value of the field WRITE TO OPER of the DSNTIPA installation panel determines whether the request is received. If the value is YES, the request is preceded by a WTOR (message number DSNJ008E) informing the operator to prepare an archive log data set for allocating.

The operator need not respond to message DSNJ008E immediately. However, delaying the response delays the offload process. It does not affect DB2 performance unless the operator delays response for so long that DB2 runs out of active logs.

The operator can respond by canceling the offload. In that case, if the allocation is for the first copy of dual archive data sets, the offload is merely delayed until the next active log data set becomes full. If the allocation is for the second copy, the archive process switches to single copy mode, but for the one data set only.

***Messages returned during offloading:*** The following messages are sent to the MVS console by DB2 and the offload process. With the exception of the DSNJ139I message, these messages can be used to find the RBA ranges in the various log data sets.

- The following message appears during DB2 initialization when the current active log data set is found, and after a data set switch. During initialization, the STARTRBA value in the message does not refer to the beginning of the data set, but to the position in the log where logging will begin.

  ```
  DSNJ001I - csect-name CURRENT COPY n ACTIVE LOG DATA SET IS
             DSNAME=...,  STARTRBA=..., ENDRBA=...
  ```

- The following message appears when an active data set is full:

  ```
  DSNJ002I - FULL ACTIVE LOG DATA SET DSNAME=...,
             STARTRBA=..., ENDRBA=...
  ```

- The following message appears when offload reaches end-of-volume or end-of-data-set in an archive log data set:

  Non-data sharing version is:

  ```
  DSNJ003I - FULL ARCHIVE LOG VOLUME DSNAME=..., STARTRBA=..., ENDRBA=...,
             STARTTIME=..., ENDTIME=..., UNIT=..., COPYnVOL=...,
             VOLSPAN=..., CATLG=...
  ```

  Data sharing version is:

  ```
  DSNJ003I - FULL ARCHIVE LOG VOLUME DSNAME=..., STARTRBA=..., ENDRBA=...,
             STARTLRSN=..., ENDLRSN=..., UNIT=..., COPYnVOL=...,
             VOLSPAN=..., CATLG=...
  ```

- The following message appears when one data set of the next pair of active logs is not available because of a delay in offloading, and logging continues on one copy only:

  ```
  DSNJ004I - ACTIVE LOG COPY n INACTIVE, LOG IN SINGLE MODE,
             ENDRBA=...
  ```

- The following message appears when dual active logging resumes after logging has been carried on with one copy only:

  ```
  DSNJ005I - ACTIVE LOG COPY n IS ACTIVE, LOG IN DUAL MODE,
             STARTRBA=...
  ```

- The following message indicates that the offload task has ended:

  ```
  DSNJ139I LOG OFFLOAD TASK ENDED
  ```

***Interruptions and errors while offloading:*** Here is how DB2 handles the following interruptions in the offloading process:

- The command STOP DB2 does not take effect until offloading is finished.

- A DB2 failure during offload causes offload to begin again from the previous start RBA when DB2 is restarted.
- Offload handling of read I/O errors on the active log is described under “Active log failure” on page 423, or write I/O errors on the archive log, under “Archive log failure” on page 427.
- An unknown problem that causes the offload task to *hang* means that DB2 cannot continue processing the log. This problem might be resolved by retrying the offload, which you can do by using the option CANCEL OFFLOAD of the command ARCHIVE LOG, described in “Canceling log off-loads” on page 340.

## Archive log data sets

Archive log data sets can be placed on standard label tapes or disks and can be managed by DFSMShsm (Data Facility Hierarchical Storage Manager). They are always written by QSAM. Archive logs on tape are read by BSAM; those on disk are read by BDAM. Each MVS logical record in an archive log data set is a VSAM CI from the active log data set. The block size is a multiple of 4 KB. (For more information, see installation panel DSNTIPA in Part 2 of *DB2 Installation Guide*.)

Output archive log data sets are dynamically allocated, with names chosen by DB2. The data set name prefix, block size, unit name, and disk sizes needed for allocation are specified when DB2 is installed, and recorded in the DSNZP*xxx* module. You can also choose, at installation time, to have DB2 add a date and time to the archive log data set name. See installation panel DSNTIPH in Part 2 of *DB2 Installation Guide* for more information.

It is not possible to specify specific volumes for new archive logs. If allocation errors occur, offloading is postponed until the next time offloading is triggered.

***Using dual archive logging:*** If you specify dual archive logs at installation time, each log CI retrieved from the active log is written to two archive log data sets. The log records that are contained on a pair of dual archive log data sets are identical, but end-of-volumes are not synchronized for multivolume data sets.

Archiving to disk offers faster recoverability but is more expensive than archiving to tape. If you use dual logging, you can specify on installation panel DSNTIPA that the primary copy of the archive log go to disk and the secondary copy go to tape.

This feature increases recovery speed without using as much disk. The second tape is intended as a backup or can be sent to a remote site in preparation for disaster recovery. To make recovering from the COPY2 archive tape faster at the remote site, use the new installation parameter, ARC2FRST, to specify that COPY2 archive log should be read first. Otherwise, DB2 always attempts to read the primary copy of the archive log data set first.

***Archiving to tape:*** If the unit name reflects a tape device, DB2 can extend to a maximum of twenty volumes. DB2 passes a file sequence number of 1 on the catalog request for the first file on the next volume. Though that might appear to be an error in the integrated catalog facility catalog, it causes no problems in DB2 processing.

If you choose to offload to tape, consider adjusting the size of your active log data sets such that each set contains the amount of space that can be stored on a nearly full tape volume. That adjustment minimizes tape handling and volume mounts and maximizes the use of tape resources. However, such an adjustment is not always necessary.

If you want the active log data set to fit on one tape volume, consider placing a copy of the BSDS on the same tape volume as the copy of the active log data set. Adjust the size of the active log data set downward to offset the space required for the BSDS.

***Archiving to disk volumes:*** All archive log data sets allocated on disk must be cataloged. If you choose to archive to disk, then the field CATALOG DATA of installation panel DSNTIPA must contain YES. If this field contains NO, and you decide to place archive log data sets on disk, you receive message DSNJ072E each time an archive log data set is allocated, although the DB2 subsystem still catalogs the data set.

If you use disk storage, be sure that the primary and secondary space quantities and block size and allocation unit are large enough so that the disk archive log data set does not attempt to extend beyond 15 volumes. That minimizes the possibility of unwanted MVS B37 or E37 abends during the offload process. Primary space allocation is set with the PRIMARY QUANTITY field of the DSNTIPA installation
# panel. The primary space quantity must be less than 64K tracks because of the
# DFSMS Direct Access Device Space Management limit of 64K tracks on a single
# volume when allocating a sequential disk data set.

***Using SMS to archive log data sets:*** If you have DFSMS/MVS (Data Facility Storage Management Subsystem) installed, it is possible to write an ACS user exit filter for your archive log data sets. Such a filter, for example, can route your output to a disk data set, which in turn can be managed by DFSMS. Be careful using an ACS filter in this manner with archive log data sets to be managed by SMS. Because SMS requires disk data sets to be cataloged, you must make sure the field CATALOG DATA on installation panel DSNTIPA contains YES. Even if it does not, message DSNJ072E is returned and the data set is forced to be cataloged by DB2.

DB2 uses the basic direct access method (BDAM) to read archive logs from disk. DFSMS/MVS does not support reading of compressed data sets using BDAM. You should not, therefore, use DFSMS/MVS hardware compression on your archive log data sets.

Ensure that DFSMS/MVS does not alter the LRECL or BLKSIZE of the archive log data sets. Altering these attributes could result in read errors when DB2 attempts to access the log data.

## Controlling the log

You can control and monitor log activity through several commands and a utility as discussed in the following sections:

“Archiving the log”
“Changing the checkpoint frequency dynamically” on page 340
“Setting limits for archive log tape units” on page 340
“Displaying log information” on page 340

## Archiving the log

**General-use Programming Interface**

A properly authorized operator can archive the current DB2 active log data sets, whenever required, by issuing the ARCHIVE LOG command. Using ARCHIVE LOG can help with diagnosis by allowing you to quickly offload the active log to the archive log where you can use DSN1LOGP to further analyze the problem.

To issue this command, you must have either SYSADM authority, or have been granted the ARCHIVE privilege.

```
-ARCHIVE LOG
```

When you issue the above command, DB2 truncates the current active log data sets, then runs an asynchronous offload, and updates the BSDS with a record of the offload. The RBA that is recorded in the BSDS is the beginning of the last complete log record written in the active log data set being truncated.

You could use the ARCHIVE LOG command as follows to capture a point of consistency for the MSTR01 and XUSR17 databases:

```
-STOP DATABASE (MSTR01,XUSR17)
-ARCHIVE LOG
-START DATABASE (MSTR01,XUSR17)
```

In this simple example, the STOP command stops activity for the databases before archiving the log.

***Quiescing activity before offloading:*** Another method of ensuring that activity has stopped before the log is archived is the MODE(QUIESCE) option of ARCHIVE LOG. With this option, DB2 users are quiesced after a commit point, and the resulting point of consistency is captured in the current active log before it is offloaded. Unlike the QUIESCE utility, ARCHIVE LOG MODE(QUIESCE) does not force all changed buffers to be written to disk and does not record the log RBA in SYSIBM.SYSCOPY. It does record the log RBA in the boot strap data set.

Consider using MODE(QUIESCE) when planning for offsite recovery. It creates a system-wide point of consistency, which can minimize the number of data inconsistencies when the archive log is used with the most current image copy during recovery.

In a data sharing group, ARCHIVE LOG MODE(QUIESCE) might result in a delay before activity on all members has stopped. If this delay is unacceptable to you, consider using ARCHIVE LOG SCOPE(GROUP) instead. This command causes truncation and offload of the logs for each active member of a data sharing group. Although the resulting archive log data sets do not reflect a point of consistency, all the archive logs are made at nearly the same time and have similar LRSN values in their last log records. When you use this set of archive logs to recover the data sharing group, you can use the ENDLRSN option in the CRESTART statement of the change log inventory utility (DSNJU003) to truncate all the logs in the group to the same point in time. See *DB2 Data Sharing: Planning and Administration* for more information.

The MODE(QUIESCE) option suspends all new update activity on DB2 up to the maximum period of time specified on the installation panel DSNTIPA, described in Part 2 of *DB2 Installation Guide*. If the time needed to quiesce is less than the time specified, then the command completes successfully; otherwise, the command fails when the time period expires. This time amount can be overridden when you issue the command, by using the TIME option:

```
-ARCHIVE LOG MODE(QUIESCE) TIME(60)
```

The above command allows for a quiesce period of up to 60 seconds before archive log processing occurs.

> **Important**
>
> Use of this option during prime time, or when time is critical, can cause a significant disruption in DB2 availability for all jobs and users that use DB2 resources.

By default, the command is processed asynchronously from the time you submit the command. (To process the command synchronously with other DB2 commands, use the WAIT(YES) option with QUIESCE; then, the MVS console is locked from DB2 command input for the entire QUIESCE period.)

During the quiesce period:

- Jobs and users on DB2 are allowed to go through commit processing, but are suspended if they try to update any DB2 resource after the commit.
- Jobs and users that only read data can be affected, because they can be waiting for locks held by jobs or users that were suspended.
- New tasks can start, but they are not allowed to update data.

As shown in the following example, the DISPLAY THREAD output issues message DSNV400I to indicate that a quiesce is in effect:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV400I - ARCHIVE LOG QUIESCE CURRENTLY ACTIVE
DSNV402I - ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID   TOKEN
BATCH    T  *    20 TEPJOB       SYSADM   DSNTEP3  0012      12
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

When all updates are quiesced, the quiesce history record in the BSDS is updated with the date and time that the active log data sets were truncated, and with the last-written RBA in the current active log data sets. DB2 truncates the current active log data sets, switches to the next available active log data sets, and issues message DSNJ311E, stating that offload started.

If updates cannot be quiesced before the quiesce period expires, DB2 issues message DSNJ317I, and archive log processing terminates. The current active log data sets are not truncated and not switched to the next available log data sets, and offload is *not* started.

Whether the quiesce was successful or not, all suspended users and jobs are then resumed, and DB2 issues message DSNJ312I, stating that the quiesce is ended and update activity is resumed.

If ARCHIVE LOG is issued when the current active log is the last available active log data set, the command is not processed, and DB2 issues this message:

```
DSNJ319I - csect-name CURRENT ACTIVE LOG DATA SET IS THE LAST
           AVAILABLE ACTIVE LOG DATA SET.  ARCHIVE LOG PROCESSING WILL
           BE TERMINATED.
```

If ARCHIVE LOG is issued when another ARCHIVE LOG command is already in progress, the new command is not processed, and DB2 issues this message:

```
DSNJ318I - ARCHIVE LOG COMMAND ALREADY IN PROGRESS.
```

***Canceling log offloads:*** It is possible for the offload of an active log to be suspended when something goes wrong with the offload process, such as a problem with allocation or tape mounting. If the active logs cannot be offloaded, DB2's active log data sets fill up and DB2 stops logging.

To avoid this problem, use the following command to cancel (and retry) an offload:

```
-ARCHIVE LOG CANCEL OFFLOAD
```

When you enter the command, DB2 restarts the offload again, beginning with the oldest active log data set and proceeding through all active log data sets that need offloading. If the offload fails again, you must fix the problem that is causing the failure before the command can work.

**End of General-use Programming Interface**

## Changing the checkpoint frequency dynamically

Use the LOGLOAD or CHKTIME option of the SET LOG command to dynamically change the checkpoint frequency without recycling DB2. The LOGLOAD value specifies the number of log records that DB2 writes between checkpoints. The CHKTIME value specifies the number of minutes between checkpoints. Either value affects the restart time for DB2.

For example, during prime shift, your DB2 shop might have a low logging rate, but require that DB2 restart quickly if it terminates abnormally. To meet this restart requirement, you can decrease the LOGLOAD value to force a higher checkpoint frequency. In addition, during off-shift hours the logging rate might increase as batch updates are processed, but the restart time for DB2 might not be as critical. In that case, you can increase the LOGLOAD value which lowers the checkpoint frequency.

You can also use the LOGLOAD or CHKTIME option to initiate an immediate system checkpoint:

```
-SET LOG LOGLOAD(0)
or
-SET LOG CHKTIME(0)
```

The CHKFREQ value that is altered by the SET LOG command persists only while DB2 is active. On restart, DB2 uses the CHKFREQ value in the DB2 subsystem parameter load module. See Chapter 2 of *DB2 Command Reference* for detailed information about this command.

## Setting limits for archive log tape units

Use the DB2 command SET ARCHIVE to set the upper limit for the number of and the deallocation time of tape units for the archive log. This command overrules the values specified during installation or in a previous invocation of the SET ARCHIVE command. The changes initiated by SET ARCHIVE are temporary; at restart, DB2 uses the values that were set during installation. See Chapter 2 of *DB2 Command Reference* for detailed information about this command.

## Displaying log information

Use the DISPLAY LOG command to display the current checkpoint frequency (either the number of log records or the minutes between checkpoints). See Chapter 2 of *DB2 Command Reference* for more details about the DISPLAY LOG and SET LOG commands.

You can obtain additional information about log data sets and checkpoints from the Print Log Map utility (DSNJU004). See Part 3 of *DB2 Utility Guide and Reference* for more information about utility DSNJU004.

## Managing the bootstrap data set (BSDS)

The BSDS is a VSAM key-sequenced data set that contains information about the log data sets and the records those data sets include. It also contains information about buffer pool attributes. The BSDS is defined with access method services when DB2 is installed and is allocated by a DD statement in the DB2 startup procedure. It is deallocated when DB2 terminates.

Normally, DB2 keeps duplicate copies of the BSDS. If an I/O error occurs, DB2 deallocates the failing copy and continues with a single BSDS. However, you can restore the dual mode as follows:

1. Use access method services to rename or delete the failing BSDS.
2. Define a new BSDS with the same name as the deleted BSDS.
3. Issue the DB2 command RECOVER BSDS to make a copy of the good BSDS in the newly allocated data set.

The active logs are first registered in the BSDS by job DSNTIJID, when DB2 is installed. They cannot be replaced, nor new ones added, without terminating and restarting DB2.

Archive log data sets are dynamically allocated. When one is allocated, the data set name is registered in the BSDS in separate entries for each volume on which the archive log resides. The list of archive log data sets expands as archives are added, and wraps around when a user-determined number of entries has been reached. The maximum number of entries is 1000 for single archive logging and 2000 for dual logging.

The inventory of archive log data sets can be managed by use of the change log inventory utility (DSNJU003). For further information, see “Changing the BSDS log inventory” on page 342.

A wide variety of tape management systems exist, along with the opportunity for external manual overrides of retention periods. Because of that, DB2 does not have an automated method to delete the archive log data sets from the BSDS inventory of archive log data sets. Thus, the information about an archive log data set can be in the BSDS long after the archive log data set has been scratched by a tape management system following the expiration of the data set’s retention period.

Conversely, the maximum number of archive log data sets could have been exceeded, and the data from the BSDS dropped long before the data set has reached its expiration date. For additional information, refer to “Deleting archive log data sets or tapes automatically” on page 343.

If you specified at installation that archive log data sets are cataloged when allocated, the BSDS points to the integrated catalog facility catalog for the information needed for later allocations. Otherwise, the BSDS entries for each volume register the volume serial number and unit information that is needed for later allocation.

## BSDS copies with archive log data sets

Each time a new archive log data set is created, a copy of the BSDS is also created. If the archive log is on tape, the BSDS is the first file on the first output volume. If the archive log is on disk, the BSDS copy is a separate file which could reside on a separate volume.

For better offload performance and space utilization, it is recommended that you use the default archive block size of 28672. If required, this value can be changed in the BLOCK SIZE field on installation panel DSNTIPA. The PRIMARY QUANTITY and SECONDARY QUANTITY fields should also be adjusted to reflect any changes in block size.

The data set names of the BSDS copy and the archive log are the same, except that the first character of the last data set name qualifier in the BSDS name is B instead of A, as in the example below:

**Archive log name**
DSNCAT.ARCHLOG1.**A**0000001

**BSDS copy name**
DSNCAT.ARCHLOG1.**B**0000001

If there is a read error while copying the BSDS, the copy is not created. Message DSNJ125I is issued, and the offload to the new archive log data set continues without the BSDS copy.

The utility DSNJU004, print log map, lists the information stored in the BSDS. For instructions on using it, see Part 3 of *DB2 Utility Guide and Reference*.

## Changing the BSDS log inventory

You do not have to take special steps to keep the BSDS updated with records of logging events—DB2 does that automatically. But you might want to change the BSDS if you:

- Add more active log data sets
- Copy active log data sets to newly allocated data sets, as when providing larger active log allocations
- Move log data sets to other devices
- Recover a damaged BSDS
- Discard outdated archive log data sets
- Create or cancel control records for conditional restart
- Add to or change the DDF communication record.

You can change the BSDS by running the DB2 batch change log inventory (DSNJU003) utility. This utility should not be run when DB2 is active. If it is run when DB2 is active, inconsistent results can be obtained. For instructions on how to use the change log inventory utility, see Part 3 of *DB2 Utility Guide and Reference*.

You can copy an active log data set using the access method services IDCAMS REPRO statement. The copy can only be performed when DB2 is down, because DB2 allocates the active log data sets as exclusive (DISP=OLD) at DB2 startup. For more information on the REPRO statement, see *DFSMS/MVS: Access Method Services for the Integrated Catalog* and *DFSMS/MVS: Access Method Services for VSAM Catalogs*.

## Discarding archive log records

You must keep enough log records to recover units of work and databases.

To recover units of recovery, you need log records at least until all current actions are completed. If DB2 abends, restart requires all log records since the previous checkpoint or the beginning of the oldest UR that was active at the abend, whichever is first on the log.

To tell whether all units of recovery are complete, read the status counts in the DB2 restart messages (shown in “Starting DB2” on page 256). If all counts are zero, no unit of recovery actions are pending. If there are indoubt units of recovery remaining, identify and recover them by the methods described in “Chapter 17. Monitoring and controlling DB2 and its connections” on page 267.

To recover databases, you need log records and image copies of table spaces. How long you keep log records depends, then, on how often you make those image copies. “Chapter 21. Backing up and recovering databases” on page 373 gives suggestions about recovery cycles; the following sections assume that you know what records you want to keep and describe only how to delete the records you do not want.

## Deleting archive log data sets or tapes automatically

You can use a disk or tape management system to delete archive log data sets or tapes automatically. The length of the retention period (in days), which is passed to the management system in the JCL parameter RETPD, is determined by the RETENTION PERIOD field on the DSNTIPA installation panel, discussed further in Part 2 of *DB2 Installation Guide*.

The default for the retention period keeps archive logs forever. If you use any other retention period, it must be long enough to contain as many recovery cycles as you plan for. For example, if your operating procedures call for a full image copy every sixty days of the least-frequently-copied table space, and you want to keep two complete image copy cycles on hand at all times, then you need an archive log retention period of at least 120 days. For more than two cycles, you need a correspondingly longer retention period.

If archive log data sets or tapes are deleted automatically, the operation does not update the archive log data set inventory in the BSDS. If you wish, you can update the BSDS with the change log inventory utility, as described in “Changing the BSDS log inventory” on page 342. The update is not really necessary; it wastes space in the BSDS to record old archive logs, but does no other harm as the archive log data set inventory *wraps* and automatically deletes the oldest entries. See “Managing the bootstrap data set (BSDS)” on page 341 for more details.

## Locating archive log data sets to delete

You must keep all the logs since the most recent checkpoint of DB2, so that DB2 can restart. You also must keep all the logs for two or more complete image copy cycles of your least-frequently-copied table space. What, then, can you discard?

You need an answer in terms of the log RBA ranges of the archive data sets. The earliest log record you want is identified by a log RBA. You can discard any archive log data sets that contain only records with log RBAs less than that.

The procedure that follows locates those data sets:

***Step 1: Resolve indoubt units of recovery:*** If DB2 is running with TSO, continue with “Find the startup log RBA”. If DB2 is running with IMS, CICS, or distributed data, the following procedure applies:

1. The period between one startup and the next must be free of any indoubt units of recovery. Ensure that no DB2 activity is going on until you finish this procedure. (You might plan this procedure for a non-prime shift, for minimum impact on users.) To find out whether indoubt units exist, issue the DB2 command DISPLAY THREAD TYPE(INDOUBT). If there are none, skip to “Find the startup log RBA”.
2. If there are one or more indoubt units of recovery, do one of the following:
   - Start IMS or CICS, causing that subsystem to resolve the indoubt units of recovery. If the thread is a distributed indoubt unit of recovery, restart the distributed data facility (DDF) to resolve the unit of work. If DDF does not start or cannot resolve the unit of work, use the command RECOVER INDOUBT to resolve the unit of work.
   - Issue the DB2 command RECOVER INDOUBT.

   These topics, including making the proper commit or abort decision, are covered in greater detail in “Resolving indoubt units of recovery” on page 363.
3. Re-issue the command DISPLAY THREAD TYPE(INDOUBT) to ensure that the indoubt units have been recovered. When none remain, continue with “Find the startup log RBA”.

***Step 2: Find the startup log RBA:*** Keep at least all log records with log RBAs greater than the one given in this message, issued at restart:

```
DSNR003I RESTART...PRIOR CHECKPOINT RBA=xxxxxxxxxxxx
```

If you suspended DB2 activity while performing step 1, you can restart it now.

***Step 3: Find the minimum log RBA needed:*** Suppose that you have determined to keep some number of complete image copy cycles of your least-frequently-copied table space. You now need to find the log RBA of the earliest full image copy you want to keep.

1. If you have any table spaces so recently created that no full image copies of them have ever been taken, take full image copies of them. If you do not take image copies of them, and you discard the archive logs that log their creation, DB2 can never recover them.

**General-use Programming Interface**

The following SQL statement lists table spaces that have no full image copy:

```
SELECT X.DBNAME, X.NAME, X.CREATOR, X.NTABLES, X.PARTITIONS
  FROM SYSIBM.SYSTABLESPACE X
    WHERE NOT EXISTS (SELECT * FROM SYSIBM.SYSCOPY Y
                        WHERE X.NAME = Y.TSNAME
                        AND X.DBNAME = Y.DBNAME
                        AND Y.ICTYPE = 'F')
  ORDER BY 1, 3, 2;
```

**End of General-use Programming Interface**

2. Issue the following SQL statement to find START_RBA values:

**General-use Programming Interface**

```
SELECT DBNAME, TSNAME, DSNUM, ICTYPE, ICDATE, HEX(START_RBA)
  FROM SYSIBM.SYSCOPY
  ORDER BY DBNAME, TSNAME, DSNUM, ICDATE;
```

**End of General-use Programming Interface**

   The statement lists all databases and table spaces within them, in ascending order by date.

   Find the START_RBA for the earliest full image copy (ICTYPE=F) that you intend to keep. If your least-frequently-copied table space is partitioned, and you take full image copies by partition, use the earliest date for all the partitions.

   If you are going to discard records from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX, note the date of the earliest image copy you want to keep.

***Step 4: Copy catalog and directory tables:*** Take full image copies of the DB2 table spaces listed below, to ensure that copies of them are included in the range of log records you will keep.

| Database name | Table space names | |
|---|---|---|
| DSNDB01 | DBD01<br>SCT02<br>SPT01 | SYSUTILX<br>SYSLGRNX |
| DSNDB06 | SYSCOPY<br>SYSDBASE<br>SYSDBAUT<br>SYSGPAUT<br>SYSGROUP<br>SYSPKAGE | SYSPLAN<br>SYSSTATS<br>SYSSTR<br>SYSUSER<br>SYSVIEWS |

***Step 5: Locate and discard archive log volumes:*** Now that you know the minimum LOGRBA, from step 3, suppose that you want to find archive log volumes that contain only log records earlier than that. Proceed as follows:

1. Execute the print log map utility to print the contents of the BSDS. For an example of the output, see the description of print log map (DSNJU004) in Part 3 of *DB2 Utility Guide and Reference*.
2. Find the sections of the output titled “ARCHIVE LOG COPY n DATA SETS”. (If you use dual logging, there are two sections.) The columns labelled STARTRBA and ENDRBA show the range of log RBAs contained in each volume. Find the volumes (two, for dual logging) whose ranges include the minimum log RBA you found in step 3; these are the earliest volumes you need to keep.

   If no volumes have an appropriate range, one of these cases applies:

   - The minimum LOGRBA has not yet been archived, and you can discard all archive log volumes.
   - The list of archive log volumes in the BSDS wrapped around when the number of volumes exceeded the number allowed by the RECORDING MAX field of installation panel DSNTIPA. If the BSDS does not register an archive log volume, it can never be used for recovery. Therefore, you should consider

adding information about existing volumes to the BSDS. For instructions, see Part 3 of *DB2 Utility Guide and Reference*.

You should also consider increasing the value of MAXARCH. For information, see information about installation panel DSNTIPA in Part 2 of *DB2 Installation Guide*.

3. Delete any archive log data set or volume (both copies, for dual logging) whose ENDRBA value is less than the STARTRBA value of the earliest volume you want to keep.

   Because BSDS entries wrap around, the first few entries in the BSDS archive log section might be more recent than the entries at the bottom. Look at the combination of date and time to compare age. Do not assume you can discard all entries *above* the entry for the archive log containing the minimum LOGRBA.

   Delete the data sets. If the archives are on tape, scratch the tapes; if they are on disks, run an MVS utility to delete each data set. Then, if you want the BSDS to list only existing archive volumes, use the change log inventory utility to delete entries for the discarded volumes; for an example, see Part 3 of *DB2 Utility Guide and Reference*.

# Chapter 19. Restarting DB2 after termination

This chapter tells what to expect when DB2 terminates normally or abnormally, and how to start it again. The concepts are important background for “Chapter 20. Maintaining consistency across multiple systems” on page 359 and “Chapter 21. Backing up and recovering databases” on page 373. This chapter includes the following topics:

“Chapter 20. Maintaining consistency across multiple systems” on page 359 describes additional considerations for a DB2 subsystem that must be kept consistent with some other system. The term *object*, used throughout this chapter, refers to any database, table space, or index space.

***Restarting in a data sharing environment:*** In a data sharing environment, restart processing is expanded to handle the coordination of data recovery across more than one DB2 subsystem. When certain critical resources are lost, restart includes additional processing to recovery and rebuild those resources. This process is called *group restart*, which is described in Chapter 5 of *DB2 Data Sharing: Planning and Administration*.

## Termination

DB2 terminates normally in response to the command STOP DB2. If DB2 stops for any other reason, the termination is considered abnormal.

## Normal termination

In a normal termination, DB2 stops all activity in an orderly way. You can use either STOP DB2 MODE (QUIESCE) or STOP DB2 MODE (FORCE). The effects are given in Table 66.

*Table 66. Termination using QUIESCE and FORCE*

| Thread type | QUIESCE | FORCE |
|---|---|---|
| Active threads | Run to completion | Roll back |
| New threads | Permitted | Not permitted |
| New connections | Not permitted | Not permitted |

You can use either command to prevent new applications from connecting to DB2.

When you give the command STOP DB2 MODE(QUIESCE), current threads can run to completion, and new threads can be allocated to an application that is running.

With IMS and CICS, STOP DB2 MODE(QUIESCE) allows a current thread to run only to the end of the unit of recovery, unless either of the following conditions are true:
- There are open, held cursors.
- Special registers are not in their original state.

Before DB2 can come down, all held cursors must be closed and all special registers must be in their original state, or the transaction must complete.

With CICS, QUIESCE mode brings down the CICS attachment facility, so an active task will not necessarily run to completion.

For example, assume that a CICS transaction opens no cursors declared WITH HOLD and modifies no special registers. The transaction does the following:

```
EXEC SQL
 ⋮          ← -STOP DB2 MODE(QUIESCE) issued here

SYNCPOINT
 ⋮

EXEC SQL    ←  This receives an AETA abend
```

The thread is allowed only to run through the first SYNCPOINT.

When you give the command STOP DB2 MODE(FORCE), no new threads are allocated, and work on existing threads is rolled back.

During shutdown, use the command DISPLAY THREAD to check its progress. If shutdown is taking too long, you can issue STOP DB2 MODE (FORCE), but rolling back work can take as much or more time as the completion of QUIESCE.

When stopping in either mode, the following steps occur:
1. Connections end.
2. DB2 ceases to accept commands.
3. DB2 disconnects from the IRLM.
4. The shutdown checkpoint is taken and the BSDS is updated.

A data object could be left in an inconsistent state, even after a shutdown with mode QUIESCE, if it was made unavailable by the command STOP DATABASE, or if DB2 recognized a problem with the object. MODE (QUIESCE) does not wait for asynchronous tasks that are not associated with any thread to complete, before it stops DB2. This can result in data commands such as STOP DATABASE and START DATABASE having outstanding units of recovery when DB2 stops. These will become inflight units of recovery when DB2 is restarted, then be returned to their original states.

## Abends

An abend can leave data in an inconsistent state for any of the following reasons:
- Units of recovery might be interrupted before reaching a point of consistency.
- Committed data might not be written to external media.
- Uncommitted data might be written to external media.

## Normal restart and recovery

DB2 uses its recovery log and the bootstrap data set (BSDS) to determine what to recover when restarting. The BSDS identifies the active and archive log data sets, the location of the most recent DB2 checkpoint on the log, and the high-level qualifier of the integrated catalog facility catalog name.

After DB2 is initialized, the restart process goes through four phases, which are described in the following sections:

“Phase 1: Log initialization”
“Phase 2: Current status rebuild” on page 350
“Phase 3: Forward log recovery” on page 351
“Phase 4: Backward log recovery” on page 352

In the descriptions that follow, the terms *inflight*, *indoubt*, *in-commit*, and *in-abort* refer to statuses of a unit of work that is coordinated between DB2 and another system, such as CICS, IMS, or a remote DBMS. For definitions of those terms, see “Maintaining consistency after termination or failure” on page 361.

At the end of the fourth phase recovery, a checkpoint is taken, and committed changes are reflected in the data.

Application programs that do not commit often enough cause long running units of recovery (URs). These long running URs might be inflight after a DB2 failure. Inflight URs can extend DB2 restart time. You can restart DB2 more quickly by postponing the backout of long running URs. Use installation options LIMIT BACKOUT and BACKOUT DURATION to establish what work to delay during restart.

If your DB2 subsystem has the UR checkpoint count option enabled, DB2 generates console message DSNR035I and trace records for IFCID 0313 to inform you about long running URs. The UR checkpoint count option is enabled at installation time, through field UR CHECK FREQ on panel DSNTIPN. See Part 2 of *DB2 Installation Guide* for more information about enabling this option.

If your DB2 subsystem has the UR log threshold option enabled, DB2 generates console message DSNR031I when an inflight UR writes more than the installation-defined number of log records. DB2 also generates trace records for IFCID 0313 to inform you about these long running URs. You can enable the UR log threshold option at installation time, through field UR LOG WRITE CHECK on panel DSNTIPN. See Part 2 of *DB2 Installation Guide* for more information about enabling this option.

You can restart a large object (LOB) table space like other table spaces. LOB table spaces defined with LOG NO do not log LOB data, but they log enough control information (and follow a force-at-commit policy) so that they can restart without loss of data integrity.

## Phase 1: Log initialization

During phase 1, DB2 attempts to locate the last log RBA written before termination. Logging continues at the next log RBA after that. In phase 1, DB2:

1. Compares the high-level qualifier of the integrated catalog facility catalog name, in the BSDS, with the corresponding qualifier of the name in the current subsystem parameter module (DSNZP*xxx*).
   - If they are equal, processing continues with step 2 on page 350.
   - If they are not equal, DB2 terminates with this message:

     ```
     DSNJ130I ICF CATALOG NAME IN BSDS
              DOES NOT AGREE WITH DSNZPARM.
              BSDS CATALOG NAME=aaaaa,
              DSNZPARM CATALOG NAME=bbbbb
     ```

     Without the check, the next DB2 session could conceivably update an entirely different catalog and set of table spaces. If the check fails, you presumably have the wrong parameter module. Start DB2 with the command START DB2 PARM(*module-name*), and name the correct module.

2. Checks the consistency of the timestamps in the BSDS.
   - If both copies of the BSDS are current, DB2 tests whether the two timestamps are equal.
     - If they are equal, processing continues with step 3.
     - If they are not equal, DB2 issues message DSNJ120I and terminates. That can happen when the two copies of the BSDS are maintained on separate disk volumes (as recommended) and one of the volumes is restored while DB2 is stopped. DB2 detects the situation at restart.

       To recover, copy the BSDS with the latest timestamp to the BSDS on the restored volume. Also recover any active log data sets on the restored volume, by copying the dual copy of the active log data sets onto the restored volume. For more detailed instructions, see “BSDS failure” on page 429.
   - If one copy of the BSDS was deallocated, and logging continued with a single BSDS, a problem could arise. If both copies of the BSDS are maintained on a single volume, and the volume was restored, or if both BSDS copies were restored separately, DB2 might not detect the restoration. In that case, log records not noted in the BSDS would be unknown to the system.
3. Finds in the BSDS the log RBA of the last log record written before termination.

   The highest RBA field (as shown in the output of the print log map utility) is updated only when the following events occur:
   - When DB2 is stopped normally (-STOP DB2).
   - When active log writing is switched from one data set to another.
   - When DB2 has reached the end of the log output buffer. The size of this buffer is determined by the OUTPUT BUFFER field of installation panel DSNTIPL described in Part 2 of *DB2 Installation Guide*.
4. Scans the log forward, beginning at the log RBA of the most recent log record, up to the last control interval (CI) written before termination.
5. Prepares to continue writing log records at the next CI on the log.
6. Issues message DSNJ099I, which identifies the log RBA at which logging continues for the current DB2 session. That message signals the end of the log initialization phase of restart.

## Phase 2: Current status rebuild

During phase 2, DB2 determines the statuses of objects at the time of termination. By the end of the phase, DB2 has determined whether any units of recovery were interrupted by the termination. In phase 2, DB2:

1. Checks the BSDS to find the log RBA of the last complete checkpoint before termination.
2. Processes the RESTART or DEFER option of the parameter module of the START DB2 command if any exist. The default is always RESTART ALL.
3. Reads every log record from that checkpoint up to the end of the log (which was located during phase 1), and identifies:
   - All exception conditions existing for each database and all image copy information related to the DSNDB01.SYSUTILX, DSNDB01.DBD01, and DSNDB06.SYSCOPY table spaces.
   - All objects open at the time of termination, and how far back in the log to go to reconstruct data pages that were not written to disk.

The number of log records written between one checkpoint and the next is set when DB2 is installed; see the field CHECKPOINT FREQ of installation panel DSNTIPN, described in Part 2 of *DB2 Installation Guide*.

You can temporarily modify the checkpoint frequency by using the command SET LOG. The value you specify persists while DB2 is active; on restart, DB2 uses the value that is specified in the CHECKPOINT FREQ field of installation panel DSNTIPN. See Chapter 2 of *DB2 Command Reference* for detailed information about this command.

4. Issues message DSNR004I, which summarizes the activity required at restart for outstanding units of recovery.
5. Issues message DSNR007I if any outstanding units of recovery are discovered. The message includes, for each outstanding unit of recovery, its connection type, connection ID, correlation ID, authorization ID, plan name, status, log RBA of the beginning of the unit of recovery (URID), and the date and time of its creation.

During phase 2, no database changes are made, nor are any units of recovery completed. DB2 determines what processing is required by phase 3 forward log recovery before access to databases is allowed.

## Phase 3: Forward log recovery

During phase 3, DB2 completes the processing for all committed changes and database write operations. This includes:

- Making all database changes for each indoubt unit of recovery and locking the data to prevent access to it after restart
- Making database changes for inflight and in-abort units of recovery, and in the case of group restarts in which locks have been lost, acquiring locks to prevent access to data in use by those units of recovery

DB2 can use the fast log apply process to enhance performance of the forward recovery phase. See the Log Apply Storage field on panel DSNTIPL in *DB2 Installation Guide*, for details on defining storage for the sort used in fast log apply processing. DB2 executes these steps:

1. Detects whether a page set being recovered is at the same level ID as it was when the page set was last closed. If it is not, DB2 issues message DSNB232I and places the pages for that object on the logical page list (LPL). DB2 does not restart that object. In this case, you must recover from the down level page set using one of the methods described in “Recovery from down-level page sets” on page 435.
2. Scans the log forward, beginning at the lowest (earliest) log RBA that is either required for completion of database writes or is associated with the “Begin Unit of Recovery” of units of recovery that require locks.

   That log RBA is determined during phase 2. REDO log records for all units of recovery are processed in this phase.
3. Uses the log RBA of the earliest potential redo log record for each object (determined during phase 2). All earlier changes to the object have been written to disk; therefore, DB2 ignores their log records.
4. Reads the data or index page for each remaining redo log record. The page header registers the log RBA of the record of the last change to the page.
   - If the log RBA of the page header is greater than or equal to that of the current log record, the logged change has already been made and written to disk, and the log record is ignored.

- If the log RBA in the page header is less than that of the current log record, the change has not been made; DB2 makes the change to the page in the buffer pool.

5. Writes pages to disk as the need for buffers demands it.
6. Marks the completion of each unit of recovery processed. If restart processing terminates later, those units of recovery do not reappear in status lists.
7. Stops scanning at the current end of the log.
8. Writes to disk all modified buffers not yet written.
9. Issues message DSNR005I, which summarizes the number of remaining in-commit or indoubt units of recovery. There should not be any in-commit units of recovery, because all processing for these should have completed. The number of indoubt units of recovery should be equal to the number specified in the previous DSNR004I restart message.
10. Issues message DSNR007I (described in "Phase 2: Current status rebuild" on page 350), which identifies any outstanding unit of recovery that still must be processed.

If DB2 encounters a problem while applying log records to an object during phase 3, the affected pages are placed in the logical page list. Message DSNI001I is issued once per page set or partition, and message DSNB250E is issued once per page. Restart processing continues.

DB2 issues status message DSNR031I periodically during this phase.

## Phase 4: Backward log recovery

During phase 4, DB2 changes performed for inflight or in-abort units of recovery are reversed. In phase 4, DB2:

1. Scans the log backward, starting at the current end. The scan continues until the earliest "Begin Unit of Recovery" record for any outstanding inflight or in-abort unit of recovery.

   If you specified limited backout processing, the scan might stop prematurely (however, DB2 guarantees that all catalog and directory changes are completely backed out). In this case, the scan completes after DB2 receives the RECOVER POSTPONED command. You can have DB2 automatically issue this command after restart by specifying the AUTO option for limited backout processing, or you can issue this command manually.
2. Reads the data or index page for each remaining undo log record. The page header registers the log RBA of the record of the last change to the page.
   - If the log RBA of the page header is greater than or equal to that of the current log record, the logged change has already been made and written to disk, therefore DB2 reverses it.
   - If the log RBA in the page header is less than that of the current log record, the change has not been made; DB2 ignores it.
3. Writes redo compensation information in the log for each undo log record, as it does when backing out a unit of recovery. The redo records describe the reversal of the change and facilitate media recovery. They are written under all circumstances, even when:
   - The disk version of the data did not need to be reversed.
   - The page set has pages on the LPL.
   - An I/O error occurred on the disk version of the data.
   - The disk version of the data could not be allocated or opened.
4. Writes pages to disk as the need for buffers demands it.

5. Finally, writes to disk all modified buffers that have not yet been written.
6. Issues message DSNR006I, which summarizes the number of remaining inflight, in-abort, and postponed-abort units of recovery. The number of inflight and in-abort units of recovery should be zero; the number of postponed-abort units of recovery might not be zero.
7. Marks the completion of each completed unit of recovery in the log so that, if restart processing terminates, the unit of recovery is not processed again at the next restart.
8. If necessary, reacquires write claims for the objects on behalf of the indoubt and postponed-abort units of recovery.
9. Takes a checkpoint, after all database writes have been completed.

If DB2 encounters a problem while applying a log record to an object during phase 4, the affected pages are placed in the logical page list. Message DSNI001I is issued once per page set or partition, and message DSNB250E is issued once per page. Restart processing continues.

DB2 issues status message DSNR031I periodically during this phase.

## Restarting automatically

If you are running DB2 in a Sysplex, and on the appropriate level of MVS, you can have the automatic restart function of MVS automatically restart DB2 or IRLM after a failure.

When DB2 or IRLM stops abnormally, MVS determines whether MVS failed too, and where DB2 or IRLM should be restarted. It then restarts DB2 or IRLM.

You must have DB2 installed with a command prefix scope of S to take advantage of automatic restart. See Part 2 of *DB2 Installation Guide* for instruction on specifying command scope.

***Using an automatic restart policy:*** You control how automatic restart works by using automatic restart policies. When the automatic restart function is active, the default action is to restart the subsystems when they fail. If this default action is not what you want, then you must create a policy defining the action you want taken.

To create a policy, you need the element names of the DB2 and IRLM subsystems:

- For a non-data-sharing DB2, the element name is 'DB2$' concatenated by the subsystem name (DB2$DB2A, for example). To specify that a DB2 subsystem is not to be restarted after a failure, include RESTART_ATTEMPTS(0) in the policy for that DB2 element.
- For local mode IRLM, the element name is a concatenation of the IRLM subsystem name and the IRLM ID. For global mode IRLM, the element name is a concatenation of the IRLM data sharing group name, IRLM subsystem name, and the IRLM ID.

For instructions on defining automatic restart policies, see *OS/390 MVS Setting Up a Sysplex*.

## Deferring restart processing

Usually, restarting DB2 activates restart processing for objects that were available when DB2 terminated (in other words, not stopped with the command STOP DATABASE). Restart processing applies or backs out log records for objects that have unresolved work.

Restart processing is controlled by what you choose on installation panel DSNTIPS, and the default is RESTART ALL.

If some specific object is causing problems, you should consider deferring its restart processing by starting DB2 without allowing that object to go through restart processing. When you defer restart of an object, DB2 puts pages necessary for the object's restart in the logical page list (LPL). Only those pages are inaccessible; the rest of the object can still be accessed after restart.

There are exceptions: when you say DEFER ALL at a site that is designated as RECOVERYSITE in DSNZP*xxx,* then all pages are placed in the LPL (as a page range, not as a huge list of individual pages). Also, if any changes must be backed out for a nonpartitioning index, then all pages for the index are placed in the LPL.

DB2 can also defer restart processing for particular objects. DB2 puts pages in the LPL for any object (or specific pages of an object) with certain problems, such as an open or I/O error during restart. Again, only pages that are affected by the error are placed on the LPL.

You can defer an object's restart processing with any of the following actions:

**VARY the device (or volume) on which the objects reside OFFLINE.** If the data sets containing an object are not available, and the object requires recovery during restart, DB2 flags it as stopped and requiring deferred restart. DB2 then restarts without it.

**Delay the backout of a long running UR.** On installation panel DSNTIPN, you can use the following options:

- LIMIT BACKOUT defined as YES or AUTO indicates that some backout processing will be postponed when restarting DB2. Users must issue the RECOVER POSTPONED command to complete the backout processing when the YES option is selected. DB2 does the backout work automatically after DB2 is running and receiving new work when the AUTO option is selected.
- BACKOUT DURATION indicates the number of log records, specified as a multiplier, to be read during restart's backward log scan.

The amount of backout processing to be postponed is determined by:

- The frequency of checkpoints
- The BACKOUT DURATION installation option
- The characteristics of the inflight and in-abort activity when the system went down

Selecting a limited backout affects log processing during restart. The backward processing of the log proceeds until the oldest inflight or in-abort UR with activity against the catalog or directory is backed out, and the requested number of log records have been processed.

**Name the object with DEFER when installing DB2.** On installation panel DSNTIPS, you can use the following options:

- DEFER ALL defers restart log apply processing for all objects, including DB2 catalog and directory objects.
- DEFER *list_of_objects* defers restart processing only for objects in the list.

Alternatively, you can specify RESTART *list_of_objects*, which limits restart processing to the list of objects in the list.

DEFER does not affect processing of the log during restart. Therefore, even if you specify DEFER ALL, DB2 still processes the full range of the log for both the forward and backward log recovery phases of restart. However, logged operations are not applied to the data set.

## Restarting with conditions

If you want to skip some portion of the log processing during DB2 restart, you can use a conditional restart. However, if a conditional restart skips any database change log records, data in the associated objects becomes inconsistent and any attempt to process them for normal operations might cause unpredictable results. The only operations that can safely be performed on the objects are recovery to a prior point of consistency, total replacement, or dropping.

In unusual cases, you might choose to make inconsistent objects available for use without recovering them. For example, the only inconsistent object might be a table space that is dropped as soon as DB2 is restarted, or the DB2 subsystem might be used only for testing application programs still under development. In cases like those, where data consistency is not critical, normal recovery operations can be partially or fully bypassed by using conditional restart control records in the BSDS. The procedure is:

1. While DB2 is stopped, run the change log inventory utility using the CRESTART control statement to create a new conditional restart control record.
2. Restart DB2. The type of recovery operations that take place is governed by the current conditional restart control record.

When considering a conditional restart, it is often useful to run the DSN1LOGP utility and review a summary report of the information contained in the log.

For an example of the messages that are written to the DB2 console during restart processing, see “Messages at start” on page 256.

In a data sharing environment, you can use the new LIGHT(YES) parameter on the START DB2 command to quickly recover retained locks on a DB2 member. For more information, see *DB2 Data Sharing: Planning and Administration*.

This section gives an overview of the available options for conditional restart. For more detail, see information about the change log inventory utility (DSNJU003) in Part 3 of *DB2 Utility Guide and Reference*. For information on data sharing considerations, see Chapter 5 of *DB2 Data Sharing: Planning and Administration*.

## Resolving postponed units of recovery

You can postpone some of the backout work associated with long running units of work during system restart by using the LIMIT BACKOUT installation option. By delaying such backout work, the DB2 subsystem can be restarted more quickly. If you specify LIMIT BACKOUT = YES, then you must use the RECOVER POSTPONED command to resolve postponed units of recovery. See Part 2 of *DB2 Installation Guide* for more information about installation options.

Use the RECOVER POSTPONED command to complete postponed backout processing on *all* units of recovery; you cannot specify a single unit of work for resolution. This command might take several hours to complete depending on the content of the long-running job. In some circumstances, you can elect to use the CANCEL option of the RECOVER POSTPONED command. This option leaves the objects in an inconsistent state (REFP) that you must resolve before using the objects. However, you might choose the CANCEL option for the following reasons:

- You determine that the complete recovery of the postponed units of recovery will take more time to complete than you have available. Further, you determine it is faster to either recover the objects to a prior point in time (PIT) or run the LOAD utility with the REPLACE option.
- You want to replace the existing data in the object with new data.
- You decide to drop the object. To drop the object successfully, complete the following steps:
  1. Issue the RECOVER POSTPONED command with the CANCEL option.
  2. Issue the DROP TABLESPACE statement.
- You do not have the DB2 logs to successfully recover the postponed units of recovery.

## Errors encountered during RECOVER POSTPONED processing

If a required page cannot be accessed during RECOVER POSTPONED processing, or if any other error is encountered while attempting to apply a log record, the page set or partition is deferred and processing continues. DB2 writes a compensation log record to reflect those deferrals and places the page in the logical page list. Some errors encountered while recovering indexes cause the entire page set to be placed in the logical page list. Some errors halt the construction of the compensation log and mark the page set as RECP.

When an error prevents logging of a compensation log record, DB2 abends. If DB2 abends:

1. Fix the error.
2. Restart DB2.
3. Re-issue the RECOVER POSTPONED command if automatic backout processing has not been specified.

## Output from RECOVER POSTPONED processing

Output from the RECOVER POSTPONED command consists of informational messages. In Figure 40 on page 357, backout processing was performed against two table space partitions and two index partitions:

```
DSNV435I ! RESOLUTION OF POSTPONED ABORT URS HAS BEEN SCHEDULED
DSN9022I ! DSNVRP 'RECOVER POSTPONED' NORMAL COMPLETION
DSNI024I ! DSNIARPL BACKOUT PROCESSING HAS COMPLETED
      FOR PAGESET DSNDB04 .I PART 00000004.
DSNI024I ! DSNIARPL BACKOUT PROCESSING HAS COMPLETED
      FOR PAGESET DSNDB04 .PT PART 00000004.
DSNI024I ! DSNIARPL BACKOUT PROCESSING HAS COMPLETED
      FOR PAGESET DSNDB04 .I PART 00000002.
DSNI024I ! DSNIARPL BACKOUT PROCESSING HAS COMPLETED
      FOR PAGESET DSNDB04 .PT PART 00000002.
```

*Figure 40. Example of output from RECOVER POSTPONED processing*

## Recovery operations you can choose for conditional restart

The recovery operations that take place during restart are controlled by the currently active conditional restart control record. An active control record is created or deactivated by running the change log inventory utility with the CRESTART control statement. You can choose:

- To retain a specific portion of the log for future DB2 processing
- To read the log forward to recover indoubt and uncommitted units of recovery
- To read the log backward to back out uncommitted and in-abort units of recovery
- To do a cold start, not processing any log records.

A conditional restart record that specifies left truncation of the log causes any postponed abort units of recovery that began earlier than the truncation RBA to end without resolution. The combination of unresolved postponed abort units of recovery can cause more records than requested by the BACKODUR system parameter to be processed. The left truncation RBA takes precedence over BACKODUR in this case.

Be careful about doing a conditional restart that discards log records. If the discarded log records contain information from an image copy of the DB2 directory, a future execution of the RECOVER utility on the directory will fail. For more information, see “Recovering the catalog and directory” on page 395.

## Records associated with conditional restart

In addition to information describing the active and archive logs, the BSDS contains two queues of records associated with conditional restart.

- A wrap-around queue of conditional restart control records. Each element in the queue records the choices you made when you created the record and the progress of the restart operation it controls. When the operation is complete, the use count is set at 1 for the record and it is not used again.
- A queue of checkpoint descriptions. Because a conditional restart can specify use of a particular log record range, the recovery process cannot automatically use the most recent checkpoint. The recovery process must find the latest checkpoint within the specified range, and uses that checkpoint queue for that purpose.

Use the utility DSN1LOGP to read information about checkpoints and conditional restart control records. See Part 3 of *DB2 Utility Guide and Reference* for information about that utility.

# Chapter 20. Maintaining consistency across multiple systems

This chapter explains data consistency issues which arise when DB2 acts in conjunction with other systems, either IMS, CICS, or remote DBMSs.

The following topics are covered:

## Consistency with other systems

DB2 can work with other database management systems, including IMS, CICS, other DB2s through the distributed data facility (DDF), and other types of remote DBMSs through DDF.

If data in more than one subsystem is to be consistent, then all update operations at all subsystems for a single logical unit of work must either be committed or backed out.

### The two-phase commit process: coordinator and participant

In a distributed system, the actions of a logical unit of work might occur at more than one system. When these actions update recoverable resources, the commit process insures that either all the effects of the logical unit of work persist or that none of the effects persist, despite component, system, or communications failures.

DB2 uses a two-phase commit process in communicating between subsystems. That process is under the control of one of the subsystems, called the *coordinator*. The other systems involved are the *participants*. IMS or CICS is always the coordinator in interaction with DB2, and DB2 is always the participant. DB2 is always the coordinator in interaction with TSO and, in that case, completely controls the commit process. In interactions with other DBMSs, including other DB2s, your local DB2 can be either the coordinator or a participant.

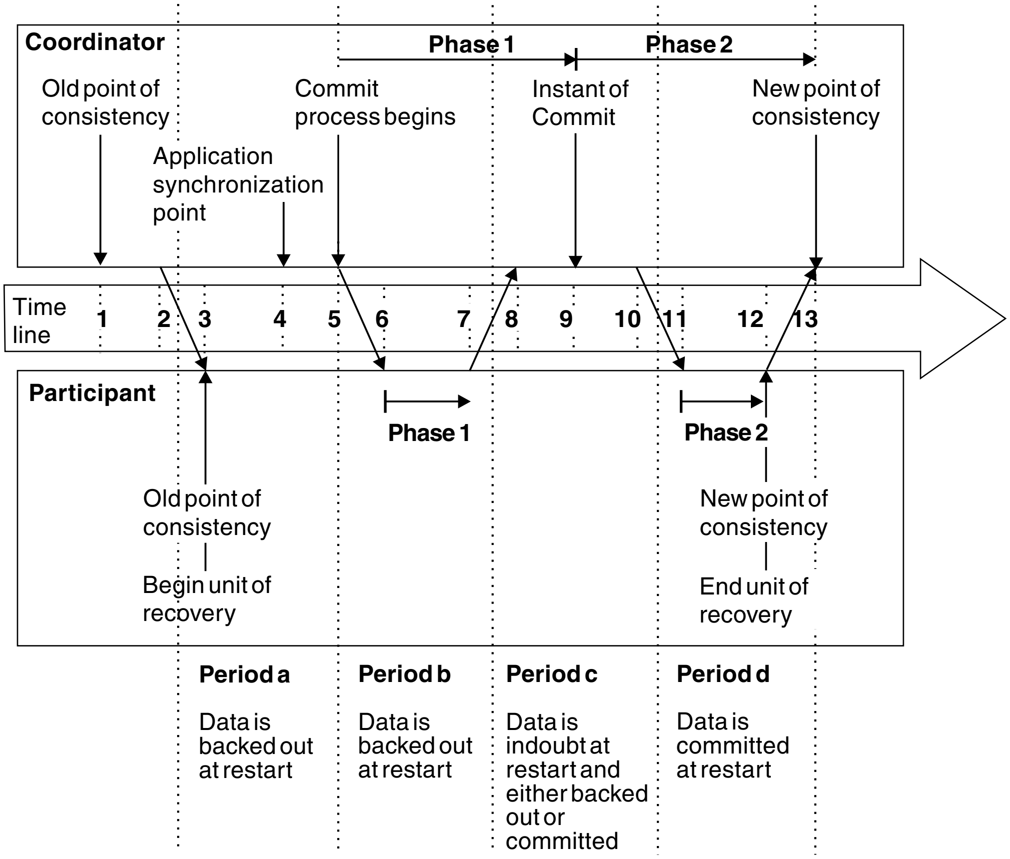


*Figure 41. Time line illustrating a commit that is coordinated with another subsystem*

## Illustration of two-phase commit

Figure 41 illustrates the two-phase commit process. Events in the coordinator (IMS, CICS, or DB2) are shown on the upper line, events in the participant on the lower line. The numbers in the following discussion are keyed to those in the figure. The resultant state of the update operations at the participant are shown between the two lines.

1. The data in the coordinator is at a point of consistency.
2. An application program in the coordinator calls the participant to update some data, by executing an SQL statement.
3. This starts a unit of recovery in the participant.
4. Processing continues in the coordinator until an application synchronization point is reached.
5. The coordinator then starts commit processing. IMS can do that by using a DL/I CHKP call, a fast path SYNC call, a GET UNIQUE call to the I/O PCB, or a normal application termination. CICS uses a SYNCPOINT command or a normal application termination. A DB2 application starts commit processing by an SQL COMMIT statement or by normal termination. Phase 1 of commit processing begins.
6. The coordinator informs the participant that it is to prepare for commit. The participant begins phase 1 processing.
7. The participant successfully completes phase 1, writes this fact in its log, and notifies the coordinator.

8. The coordinator receives the notification.
9. The coordinator successfully completes its phase 1 processing. Now both subsystems agree to commit the data changes, because both have completed phase 1 and could recover from any failure. The coordinator records on its log the instant of commit—the irrevocable decision of the two subsystems to make the changes.

   The coordinator now begins phase 2 of the processing—the actual commitment.
10. It notifies the participant to begin its phase 2.
11. The participant logs the start of phase 2.
12. Phase 2 is successfully completed, which establishes a new point of consistency for the participant. The participant then notifies the coordinator that it is finished with phase 2.
13. The coordinator finishes its phase 2 processing. The data controlled by both subsystems is now consistent and available to other applications.

There are occasions when the coordinator invokes the participant when no participant resource has been altered since the completion of the last commit process. This can happen, for example, when SYNCPOINT is issued after performance of a series of SELECT statements or when end-of-task is reached immediately after SYNCPOINT has been issued. When this occurs, the participant performs both phases of the two-phase commit during the first commit phase and records that the user or job is read-only at the participant.

## Maintaining consistency after termination or failure

If DB2 fails while acting as a coordinator, it has the appropriate information to determine commit or roll back decisions after restart. On the other hand, if DB2 fails while acting as the participant, it must determine after restart whether to commit or to roll back units of recovery that were active at the time of the failure. For certain units of recovery, DB2 has enough information to make the decision. For others, it does not, and must get information from the coordinator when the connection is reestablished.

The status of a unit of recovery after a termination or failure depends upon the moment at which the incident occurred. Figure 41 on page 360 helps to illustrate the possible statuses listed below:

**Status Description and Processing**

**Inflight**
The participant or coordinator failed before finishing phase 1 (period *a* or *b*); during restart, both systems back out the updates.

**Indoubt**
The participant failed after finishing phase 1 and before starting phase 2 (period *c*); only the coordinator knows whether the failure happened before or after the commit (point *9*). If it happened before, the participant must back out its changes; if it happened afterward, it must make its changes and commit them. After restart, the participant waits for information from the coordinator before processing this unit of recovery.

**In-commit**
The participant failed after it began its own phase 2 processing (period *d*); it makes committed changes.

**In-abort**

The participant or coordinator failed after a unit of recovery began to be rolled back but before the process was complete (not shown in the figure). The operational system rolls back the changes; the failed system continues to back out the changes after restart.

**Postponed abort**

If LIMIT BACKOUT installation option is set to YES or AUTO, any backout not completed during restart is postponed. The status of the incomplete URs is changed from inflight or in-abort to postponed abort.

## Termination

Termination for multiple systems is like termination for single systems, but with these added considerations:

- Using -STOP DB2 MODE(FORCE) could create indoubt units of recovery for threads that are between commit processing phases. They are resolved upon reconnection with the coordinator.
- Data updated by an indoubt unit of recovery is locked and unavailable for use by others. The unit could be indoubt when DB2 was stopped, or could be indoubt from an earlier termination and not yet resolved.
- A DB2 system failure can leave a unit of recovery in an indoubt state if the failure occurs between phase 1 and phase 2 of the commit process.

## Normal restart and recovery

When DB2 acts together with another system, the recovery log contains information about units of recovery that are inflight, indoubt, in-abort, postponed abort, or in-commit. The phases of restart and recovery deal with that information as follows:

### Phase 1: Log initialization

This phase proceeds as described in “Phase 1: Log initialization” on page 349.

### Phase 2: Current status rebuild

While reading the log, DB2 identifies:

- The coordinator and all participants for every unit of recovery.
- All units of recovery that are outstanding and their statuses (indoubt, in-commit, in-abort, or inflight, as described under “Maintaining consistency after termination or failure” on page 361).

### Phase 3: Forward log recovery

DB2 makes all database changes for each indoubt unit of recovery, and locks the data to prevent access to it after restart. Later, when an indoubt unit of recovery is resolved, processing is completed in one of these ways:

- For the ABORT option of the RECOVER INDOUBT command, DB2 reads and processes the log, reversing all changes.
- For the COMMIT option of the RECOVER INDOUBT command, DB2 reads the log but does not process the records because all changes have been made.

At the end of this phase, indoubt activity is reflected in the database as though the decision was made to commit the activity, but the activity has not yet been committed. The data is locked and cannot be used until DB2 recognizes and acts upon the indoubt decision. (For a description of indoubt units of recovery, see “Resolving indoubt units of recovery” on page 363.)

### Phase 4: Backward log recovery

As described in “Phase 4: Backward log recovery” on page 352, this phase reverses changes performed for inflight or in-abort units of recovery. At the end of this phase, interrupted inflight and in-abort changes have been removed from the database (the data is consistent and can be used) or removal of the changes has been postponed (the data is inconsistent and unavailable).

If removal of the changes has been postponed, the units of recovery become known as *postponed abort* units of recovery. The data with pending backout work is in a restrictive state (restart pending) which makes the data unavailable. The data becomes available upon completion of backout work or upon cold or conditional restart of DB2.

If the LIMIT BACKOUT system parameter is AUTO, completion of the backout work begins automatically by DB2 when the system accepts new work. If the LIMIT BACKOUT system parameter is YES, completion of the backout work begins when you use the RECOVER POSTPONED command.

## Restarting with conditions

If conditional restart is performed when DB2 is acting together with other systems, the following occurs:

1. All information about another coordinator and other participants known to DB2 is displayed by messages DSNL438I and DSNL439I.
2. This information is purged. Therefore the RECOVER INDOUBT command must be used at the local DB2 when the local location is a participant, and at another DB2 when the local location is the coordinator.
3. Indoubt database access threads continue to appear as indoubt and no resynchronization with either a coordinator or a participant is allowed.

Methods for resolving inconsistencies caused by conditional restart and implications in a distributed environment are described in “Resolving inconsistencies resulting from conditional restart” on page 500.

# Resolving indoubt units of recovery

If DB2 loses its connection to another system, it normally attempts to recover all inconsistent objects after restart. The information needed to resolve indoubt units of recovery must come from the coordinating system. This section describes the process of resolution for different types of other systems.

Check the console for message DSNR036I for unresolved units of recovery encountered during a checkpoint. This message might occur to remind operators of existing indoubt threads. See Part 2 of of *DB2 Installation Guide* for details.

**Important**

If the TCP/IP address associated with a DRDA server is subject to change, each DRDA server’s domain name must be defined in the CDB. This allows DB2 to recover from situations where the server’s IP address changes prior to successful resynchronization.

## Resolution of indoubt units of recovery from IMS

The resolution of indoubt units in IMS has no effect on DL/I resources. Because IMS is in control of recovery coordination, DL/I resources are never indoubt. When IMS restarts, it automatically commits or backs out incomplete DL/I work, based on whether or not the commit decision was recorded on the IMS log. The existence of indoubt units does not imply that DL/I records are locked until DB2 connects.

During the current status rebuild phase of DB2 restart, the DB2 participant makes a list of indoubt units of recovery. IMS builds its own list of residual recovery entries (RREs). The RREs are logged at IMS checkpoints until all entries are resolved.

When indoubt units are recovered, the following steps take place:

1. IMS either passes an RRE to the IMS attachment facility to resolve the entry or informs the attachment facility of a cold start. The attachment facility passes the required information to DB2.
2. If DB2 recognizes that an entry has been marked by DB2 for commit and by IMS for roll back, it issues message DSNM005I. DB2 issues this message for inconsistencies of this type between DB2 and IMS.
3. The IMS attachment facility passes a return code to IMS, indicating that it should either destroy the RRE (if it was resolved) or keep it (if it was not resolved). The procedure is repeated for each RRE.
4. Finally, if DB2 has any remaining indoubt units, the attachment facility issues message DSNM004I.

The IMS attachment facility writes all the records involved in indoubt processing to the IMS log tape as type X'5501FE'.

For all resolved units, DB2 updates databases as necessary and releases the corresponding locks. For threads that access offline databases, the resolution is logged and acted on when the database is started.

DB2 maintains locks on indoubt work that was not resolved. This can create a backlog for the system if important locks are being held. Use the DISPLAY DATABASE LOCKS command to find out which tables and table spaces are locked by indoubt units of recovery. The connection remains active so you can clean up the IMS RREs. Recover the indoubt threads by the methods described in “Controlling IMS connections” on page 295.

All indoubt work should be resolved unless there are software or operating problems, such as with an IMS cold start. Resolution of indoubt units of recovery from IMS can cause delays in SQL processing. Indoubt resolution by the IMS control region takes place at two times:

- At the start of the connection to DB2, during which resolution is done synchronously
- When a program fails, during which the resolution is done asynchronously.

In the first case, SQL processing is prevented in all dependent regions until the indoubt resolution is completed. IMS does not allow connections between IMS dependent regions and DB2 before the indoubt units are resolved.

## Resolution of indoubt units of recovery from CICS

The resolution of indoubt units has no effect on CICS resources. CICS is in control of recovery coordination and, when it restarts, automatically commits or backs out each unit, depending on whether there was or was not an end of unit of work log record. The existence of indoubt work does not lock CICS resources until DB2 connection.

A process to resolve indoubt units of recovery is initiated during start up of the attachment facility. During this process:

- The attachment facility receives a list of indoubt units of recovery for this connection ID from the DB2 participant and passes them to CICS for resolution.
- CICS compares entries from this list with entries in its own. CICS determines from its own list what action it took for the indoubt unit of recovery.
- For each entry in the list, CICS creates a task that drives the attachment facility, specifying the final commit or abort direction for the unit of recovery.
- If DB2 does not have any indoubt unit of recovery, a dummy list is passed. CICS then purges any unresolved units of recovery from previous connections, if any.

If the units of recovery cannot be resolved because of conditions described in messages DSNC001I, DSNC034I, DSNC035I, or DSNC036I, CICS enables the connection to DB2. For other conditions, it sends message DSNC016I and terminates the connection.

For all resolved units, DB2 updates databases as necessary and releases the corresponding locks. For threads that access offline databases, the resolution is logged and acted on when the database is started. Unresolved units can remain after restart; resolve them by the methods described in "Manually recovering CICS indoubt units of recovery" on page 419.

## Resolution of indoubt units of recovery between DB2 and a remote system

When communicating with a remote DBMS, indoubt units of recovery can result from failure with either the participant or coordinator or with the communication link between them even if the systems themselves have not failed.

Normally, if your subsystem fails while communicating with a remote system, you should wait until both systems and their communication link become operational. Your system then automatically recovers its indoubt units of recovery and continues normal operation. When DB2 restarts while any unit of recovery is indoubt, the data required for that unit remains locked until the unit of recovery is resolved.

If automatic recovery is not possible, DB2 alerts you to any indoubt units of recovery that you need to resolve. If it is imperative that you release locked resources and bypass the normal recovery process, you can resolve indoubt situations manually.

**Important**

In a manual recovery situation, you must determine whether the coordinator decided to commit or abort and ensure that the same decision is made at the participant. In the recovery process, DB2 attempts to automatically resynchronize with its participants. If you decide incorrectly what the coordinator's recovery action is, data is inconsistent at the coordinator and participant.

If you choose to resolve units of recovery manually, you must:

- Commit changes made by logical units of work that were committed by the other system
- Roll back changes made by logical units of work that were rolled back by the other system

### Making heuristic decisions

From DB2’s point of view, a decision to commit or roll back an indoubt unit of recovery by any means but the normal resynchronization process is a *heuristic decision*. If you commit or roll back a unit of work and your decision is different from the other system’s decision, data inconsistency occurs. This type of damage is called *heuristic damage*. If this situation should occur, and your system then updates any data involved with the previous unit of work, your data is corrupted and is extremely difficult to correct.

In order to make a correct decision, you must be absolutely sure that the action you take on indoubt units of recovery is the same as the action taken at the coordinator. Validate your decision with the administrator of the other systems involved with the logical unit of work.

### Methods for determining the coordinator’s commit or abort decision

The first step is to communicate with the other system administrator. There are several ways to ascertain the status of indoubt units at other systems:

- Use a NetView program. Write a program that analyzes NetView alerts for each involved system, and returns the results through the NetView system.
- Use an automated MVS console to ascertain the status of the indoubt threads at the other involved systems.
- Use the command DISPLAY THREAD TYPE(INDOUBT) LUWID(*luwid*).

  If the coordinator DB2 system is started and no DB2 cold start was performed, then DISPLAY THREAD TYPE(INDOUBT) can be used. If the decision was to commit, the display thread indoubt report includes the LUWID of the indoubt thread. If the decision was to abort, the thread is not displayed.
- Read the recovery log using DSN1LOGP.

  If the coordinator DB2 cannot be started, then DSN1LOGP can determine the commit decision. If the coordinator DB2 performed a cold start (or any type of conditional restart), then the system log should contain messages DSNL438I or DSNL439I, which describe the status of the unit of recovery (LUWID).

### Displaying information on indoubt threads

Use DISPLAY THREAD TYPE(INDOUBT) to find information on allied and database access indoubt threads. The command provides information about threads where DB2 is a participant, a coordinator, or both. The TYPE(INDOUBT) option tells you which systems still need indoubt resolution and provides the LUWIDs you need to recover indoubt threads. A thread that has completed phase 1 of commit and still has a connection with its coordinator is in the *prepared* state and is displayed as part of the display thread active report. If a prepared thread loses its connection with its coordinator, it enters the *indoubt* state and terminates its connections to any participants at that time. Any threads in the prepared or indoubt state when DB2 terminates are indoubt after DB2 restart. However, if the participant system is waiting for a commit or roll back decision from the coordinator, and the connection is still active, DB2 considers the thread active.

If a thread is indoubt at a participant, you can determine whether a commit or abort decision was made at the coordinator by issuing the DISPLAY THREAD command at the coordinator as described previously. If an indoubt thread appears at one system and does not appear at the other system, then the latter system backed out the thread, and the first system must therefore do the same. See “Monitoring threads” on page 283 for examples of output from the DISPLAY THREAD command.

### Recovering indoubt threads

After you determine whether you need to commit or roll back an indoubt thread, recover it with the RECOVER INDOUBT command. This command does not erase the thread’s indoubt status. It still appears as an indoubt thread until the systems go through the normal resynchronization process. An indoubt thread can be identified by its LUWID, LUNAME or IP address. You can also use the LUWID’s token with the command.

***Committing or rolling back:*** Use the ACTION(ABORT|COMMIT) option of RECOVER INDOUBT to commit or roll back a logical unit of work. If your system is the coordinator of one or more other systems involved with the logical unit of work, your action propagates to the other system associated with the LUW.

For example, to recover two indoubt threads, the first with LUWID=DB2NET.LUNSITE0.A11A7D7B2057.0002 and the second with a token of 442, and commit the LUWs, enter:

```
-RECOVER INDOUBT ACTION(COMMIT) LUWID(DB2NET.LUNSITE0.A11A7D7B2057.0002,442)
```

Detailed scenarios describing indoubt thread resolution can be found in “Resolving indoubt threads” on page 465.

### Resetting the status of an indoubt thread

Following manual recovery of an indoubt thread, allow the systems to resynchronize automatically; this resets the status of the indoubt thread. However, if heuristic damage or a protocol error occurs, you must use the RESET INDOUBT command to delete indoubt thread information for a thread whose *reset* status is *yes*. DB2 maintains this information until normal automatic recovery. You can purge information about threads where DB2 is either the coordinator or participant. If the thread is an allied thread connected with IMS or CICS, the command only applies to coordinator information about downstream participants. Information that is purged does not appear in the next display thread report and is erased from DB2’s logs.

For example, to purge information on two indoubt threads, the first with an LUWID=DB2NET.LUNSITE0.A11A7D7B2057.0002 and a resync port number of 123; and the second with a token of 442, enter:

```
-RESET INDOUBT LUWID(DB2NET.LUNSITE0.A11A7D7B2057.0002:123,442)
```

You can also use a LUNAME or IP address with the RESET INDOUBT command. A new keyword (IPADDR) can be used in place of LUNAME or LUW keywords, when the partner uses TCP/IP instead of SNA. The partner’s resync port number is required when using the IP address. The DISPLAY THREAD output lists the resync port number. This allows you to specify a location, instead of a particular thread. You can reset all the threads associated with that location with the (*) option.

## Resolution of indoubt units of recovery from OS/390 RRS

It is possible for a DB2 unit of recovery (for a thread that uses RRSAF) or for a OS/390 RRS unit of recovery (for a stored procedure) to enter the INDOUBT state. This is a state where a failure occurs when the participant (DB2 for a thread that uses RRSAF or OS/390 RRS for a stored procedure) has completed phase 1 of commit processing and is waiting for the decision from the commit coordinator. This failure could be a DB2 abnormal termination, an OS/390 RRS abnormal termination, or both.

Normally, automatic resolution of indoubt units of recovery occurs when DB2 and OS/390 RRS reestablish communication with each other. If something prevents this, then it is possible to manually resolve an indoubt unit of recovery. This process is not recommended because it might lead to inconsistencies in recoverable resources.

The following errors make manual recovery necessary:

- An OS/390 RRS cold start where the OS/390 RRS log is lost.

  If DB2 is a participant and has one or more indoubt threads, then these indoubt threads must be manually resolved in order to commit or abort the database changes and to release database locks. If DB2 is a coordinator for an OS/390 RRS unit of recovery, then DB2 knows the commit/abort decision but cannot communicate this information to the RRS compliant resource manager that has an indoubt unit of recovery.
- If DB2 performs a conditional restart and loses information from its log, then there might be inconsistent DB2 managed data.
- In a Sysplex, if DB2 is restarted on an MVS system where OS/390 RRS is not installed, then DB2 might have indoubt threads.

  This is a user error because OS/390 RRS must be started on all processors in a Sysplex on which OS/390 RRS work is to be performed.

Both DB2 and OS/390 RRS can display information about indoubt units of recovery. Both also provide techniques for manually resolving these indoubt units of recovery.

In DB2, the DISPLAY THREAD command provides information about indoubt DB2 thread. The display output includes OS/390 RRS unit of recovery IDs for those DB2 threads that have OS/390 RRS either as a coordinator or as a participant. If DB2 is a participant, the OS/390 RRS unit of recovery ID displayed can be used to determine the outcome of the OS/390 RRS unit of recovery. If DB2 is the coordinator, you can determine the outcome of the unit of recovery from the DISPLAY THREAD output.

In DB2, the RECOVER INDOUBT command lets you manually resolve a DB2 indoubt thread. You can use RECOVER INDOUBT to commit or roll back a unit of recovery after you determine what the correct decision is.

OS/390 RRS provides an ISPF interface that provides a similar capability.

## Consistency across more than two systems

The principles and methods for maintaining consistency across more than two systems are similar to those used to ensure consistency across two systems. The main difference involves a system’s role as coordinator or participant when a unit of work spans multiple systems.

## Commit coordinator and multiple participants

The coordinator of a unit of work that involves two or more other DBMSs must ensure that all systems remain consistent. After the first phase of the two-phase commit process, the DB2 coordinator waits for the other participants to indicate that they can commit the unit of work. If all systems are able, the DB2 coordinator sends the commit decision and each system commits the unit of work.

If even one system indicates that it cannot commit, then the DB2 coordinator sends out the decision to roll back the unit of work at all systems. This process ensures that data among multiple DBMSs remains consistent. When DB2 is the participant, it follows the decision of the coordinator, whether the coordinator is another DB2 or another DBMS.

DB2 is always the participant when interacting with IMS or CICS systems. However, DB2 can also serve as the coordinator for other DBMSs or DB2 subsystems in the same unit of work. For example, if DB2 receives a request from a coordinating system that also requires data manipulation on another system, DB2 propagates the unit of work to the other system and serves as the coordinator for that system.

For example, in Figure 42, DB2A is the participant for an IMS transaction, but becomes the coordinator for the two database servers (AS1 and AS2), DB2B, and its respective DB2 servers (DB2C, DB2D, and DB2E).

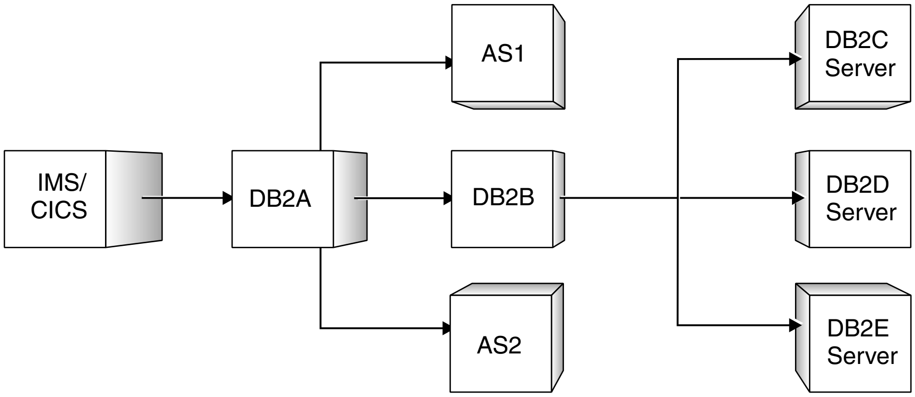


*Figure 42. Illustration of multi-site unit of work*

If the connection goes down between DB2A and the coordinating IMS system, the connection becomes an indoubt thread. However, DB2A’s connections to the other systems are still waiting and are not considered indoubt. Wait for automatic recovery to occur to resolve the indoubt thread. When the thread is recovered, the unit of work commits or rolls back and this action is propagated to the other systems involved in the unit of work.

## Illustration of multi-site update

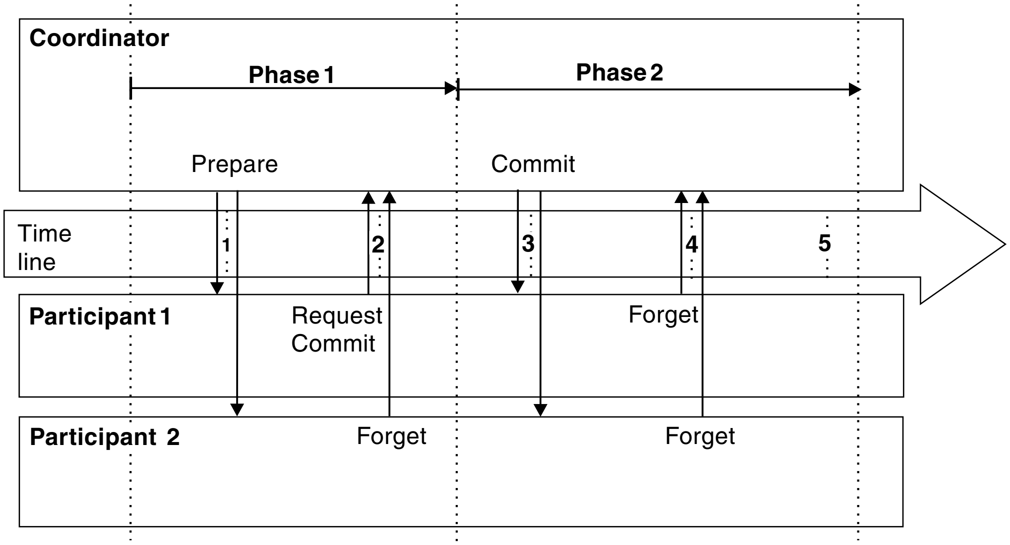


*Figure 43. Illustration of multi-site update. C is the coordinator; P1 and P2 are the participants.*

Figure 43 illustrates a multi-site update involving one coordinator and two participants.

**Phase 1:**

1. When an application commits a logical unit of work, it signals the DB2 coordinator. The coordinator starts the commit process by sending messages to the participants to determine whether they can commit.
2. A participant (*Participant 1*) that is willing to let the logical unit of work be committed, and which has updated recoverable resources, writes a log record. It then sends a request commit message to the coordinator and waits for the final decision (commit or roll back) from the coordinator. The logical unit of work at the participant is now in the prepared state.

   If a participant (*Participant 2*) has not updated recoverable resources, it sends a forget message to the coordinator, releases its locks and forgets about the logical unit of work. A read-only participant writes no log records. As far as this participant is concerned, it does not matter whether the logical unit of work ultimately gets rolled back or committed.

   If a participant wants to have the logical unit of work rolled back, it writes a log record and sends a message to the coordinator. Because a message to roll back acts like a veto, the participant in this case knows that the logical unit of work will be rolled back by the coordinator. The participant does not need any more information from the coordinator and therefore rolls back the logical unit of work, releases its locks, and forgets about the logical unit of work. (This case is not illustrated in the figure.)

**Phase 2:**

3. After the coordinator receives request commit or forget messages from all its participants, it starts the second phase of the commit process. If at least one of the responses is request commit, the coordinator writes a log record and sends committed messages to all the participants who responded to the prepare

message with request commit. If neither the participants nor the coordinator have updated any recoverable resources, there is no second phase and no log records are written by the coordinator.

4. Each participant, after receiving a committed message, writes a log record, sends a response to the coordinator, and then commits the logical unit of work.

   If any participant responds with a roll back message, the coordinator writes a log record and sends a roll back message to all participants. Each participant, after receiving a roll back message writes a log record, sends an acknowledgment to the coordinator, and then rolls back the logical unit of work. (This case is not illustrated in the figure.)

5. The coordinator, after receiving the responses from all the participants that were sent a message in the second phase, writes an ’end’ record and forgets the logical unit of work.

It is important to remember that if you try to resolve any indoubt threads manually, you need to know whether the participants committed or rolled back their units of work. With this information you can make an appropriate decision regarding processing at your site.

# Chapter 21. Backing up and recovering databases

DB2 provides means for recovering data to its current state or to an earlier state. The units of data that can be recovered are table spaces, indexes, index spaces, partitions, and data sets.

This chapter explains the following topics:

For all commands and utility statements, the complete syntax and parameter descriptions can be found in *DB2 Command Reference* and *DB2 Utility Guide and Reference*.

## Planning for backup and recovery

Development at your site of backup and recovery procedures is critical in order to avoid costly and time-consuming losses of data. You should develop procedures to:

- Create a point of consistency
- Restore system and data objects to a point of consistency
- Back up the DB2 catalog and directory and your data
- Recover the DB2 catalog and directory and your data
- Recover from out-of-space conditions
- Recover from a hardware or power failure
- Recover from an MVS component failure

In addition, you should consider a procedure for offsite recovery in case of a disaster.

To improve recovery capability in the event of a disk failure, it is advisable to use dual active logging and to place the copies of the active log data sets and the bootstrap data sets on different disk volumes. These concepts are described in “Establishing the logging environment” on page 333.

The principal tools for recovery are the utilities QUIESCE, REPORT, COPY, RECOVER, and MERGECOPY. This section also gives an overview of these utilities to help you with your backup and recovery planning.

This section covers the following topics, which you should consider when you plan for backup and recovery:

- “Considerations for recovering distributed data” on page 374
- “Considerations for recovering indexes” on page 375
- “Preparing for recovery” on page 375
- “Making backup and recovery plans that maximize availability” on page 379
- “What happens during recovery” on page 376
- “How to find recovery information” on page 382
- “Preparing to recover to a prior point of consistency” on page 383
- “Preparing to recover the entire DB2 subsystem to a prior point in time” on page 384
- “Preparing for disaster recovery” on page 385

- “Ensuring more effective recovery from inconsistency problems” on page 388
- “Running RECOVER in parallel” on page 390
- “Reading the log without RECOVER” on page 391

## Considerations for recovering distributed data

Using distributed data, no matter where a unit of work originates, the unit is processed as a whole at the target systems. At a DB2 server, the entire unit is either committed or rolled back. It is not processed if it violates referential constraints defined within the target system. Whatever changes it makes to data are logged. A set of related table spaces can be quiesced at the same point in the log, and image copies can be made of them simultaneously. If that is done, and if the log is intact, any data can be recovered after a failure and be internally consistent.

However, DB2 provides no special means to coordinate recovery between different subsystems even if an application accesses data in several systems. To guarantee consistency of data between systems, write your applications, as usual, to do all related updates within one unit of work.

Point-in-time recovery (to the last image copy or to an RBA) presents other problems. You cannot control a utility in one subsystem from another subsystem. In practice, you cannot quiesce two sets of table spaces, or make image copies of them, in two different subsystems at exactly the same instant. Neither can you recover them to exactly the same instant, because there are two different logs, and a relative byte address (RBA) does not mean the same thing for both of them.

In planning, then, the best approach is to consider carefully what the QUIESCE, COPY, and RECOVER utilities do for you and then plan not to place data that must be closely coordinated on separate subsystems. After that, recovery planning is a matter of agreement among database administrators at separate locations.

Because DB2 is responsible for recovering DB2 data only, it does not recover non-DB2 data. Non-DB2 systems do not always provide equivalent recovery capabilities.

## Extended recovery facility (XRF) toleration

DB2 can be used in an XRF recovery environment with CICS or IMS. All DB2-owned data sets (executable code, the DB2 catalog, user databases) must be on disk shared between the primary and alternate XRF processors. In the event of an XRF recovery, DB2 must be stopped on the primary processor and started on the alternate. For CICS that can be done automatically, by using the facilities provided by CICS, or manually, by the system operator. For IMS, that is a manual operation and must be done after the coordinating IMS system has completed the processor switch. In that way, any work that includes SQL can be moved to the alternate processor with the remaining non-SQL work. Other DB2 work (interactive or batch SQL and DB2 utilities) must be completed or terminated before DB2 can be switched to the alternate processor. Consider the effect of this potential interruption carefully when planning your XRF recovery scenarios.

Care must be taken to prevent DB2 from being started on the alternate processor until the DB2 system on the active, failing processor terminates. A premature start can cause severe integrity problems in data, the catalog, and the log. The use of global resource serialization (GRS) helps avoid the integrity problems by preventing simultaneous use of DB2 on the two systems. The bootstrap data set (BSDS) must be included as a protected resource, and the primary and alternate XRF processors must be included in the GRS ring.

## Considerations for recovering indexes

DB2 indexes can be recovered to currency through the REBUILD INDEX utility. The RECOVER utility recovers indexes by restoring an image copy of the index and then applying the log. The REBUILD INDEX utility reconstructs the indexes by reading the appropriate rows in the table space, extracting the index keys, sorting the keys, and then loading the index keys into the index. You can use either of these methods. For more information on the RECOVER and REBUILD INDEX utilities, see Part 2 of *DB2 Utility Guide and Reference*.

If you use the RECOVER utility for indexes, you must make several operational changes.

- You must create or alter indexes using the SQL statement ALTER INDEX with the option COPY YES before you can copy and recover them.
- You must make image copies of all indexes that you plan to recover using the RECOVER utility. The COPY utility makes full image copies or concurrent copies of indexes. Incremental copies of indexes is not supported. If full image copies of the index are taken at timely intervals, recovering a large index might be faster than rebuilding the index.
- You can recover indexes and table spaces in a single list using the RECOVER utility. By using one utility statement, the logs for all of the indexes and table spaces are processed in one pass.

The output from the DISPLAY DATABASE RESTRICT command shows the pending states for index spaces. See *DB2 Command Reference* for descriptions of status codes displayed by the DISPLAY DATABASE command.

## Preparing for recovery

The RECOVER utility supports the recovery of table spaces or index spaces. In the discussions about recovery in this chapter, the term page set can be a table space, index space, or any combination of these.

DB2 can recover a page set by using a backup copy or the recovery log or both. The DB2 recovery log contains a record of all changes made to the page set. If DB2 fails, it can recover the page set by restoring the backup copy and applying the log changes to it from the point of the backup copy.

The DB2 catalog and directory page sets must be copied at least as frequently as the most critical user page sets. Moreover, it is your responsibility to periodically copy the tables in the communications database (CDB), the application registration table, the object registration table, and the resource limit facility (governor), or to maintain the information necessary to re-create them. Plan your backup strategy accordingly.

***A recovery preparation scenario:*** The following backup scenario suggests how DB2 utilities might be used:

Imagine that you are the database administrator for DBASE1. Table space TSPACE1 in DBASE1 has been available all week. On Friday, a disk write operation for TSPACE1 fails. You need to recover the table space to the last consistent point before the failure occurred. You can do that because you have regularly followed a cycle of preparations for recovery. The most recent cycle began on Monday morning.

***Monday morning:*** You start the DBASE1 database and make a *full image copy* of TSPACE1 and all indexes immediately. That gives you a starting point from which to recover. Use the COPY utility with the SHRLEVEL CHANGE option to improve availability. See Part 2 of *DB2 Utility Guide and Reference* for more information about the COPY utility.

***Tuesday morning:*** You run COPY again. This time you make an incremental image copy to record only the changes made since the last full image copy you took on Monday. You also make a full index copy.

TSPACE1 can be accessed and updated while the image copy is being made. For maximum efficiency, however, you schedule the image copies when online use is minimal.

***Wednesday morning:*** You make another incremental image copy, and then create a full image copy by using the MERGECOPY utility to merge the incremental image copy with the full image copy.

***Thursday and Friday mornings:*** You make another incremental image copy and a full index copy each morning.

***Friday afternoon:*** An unsuccessful write operation occurs and you need to recover the table space. Run the RECOVER utility, as described in Part 2 of *DB2 Utility Guide and Reference*. The utility restores the table space from the full image copy made by MERGECOPY on Wednesday and the incremental image copies made on Thursday and Friday, and includes all changes made to the recovery log since Friday morning.

***Later Friday afternoon:*** The RECOVER utility issues a message announcing that it has successfully recovered TSPACE1 to current point-in-time.

This imaginary scenario is somewhat simplistic. You might not have taken daily incremental image copies on just the table space that failed. You might not ordinarily recover an entire table space. However, it illustrates this important point: with proper preparation, recovery from a failure is greatly simplified.

## What happens during recovery

Figure 44 on page 377 shows an overview of the recovery process. To recover a page set, the RECOVER utility typically uses these items:

- A full image copy; that is, a complete copy of the page set
- For table spaces only, any later incremental image copies; each summarizes all the changes made to the table space from the time the previous image copy was made
- All log records for the page set that were created since the most recent image copy.

The RECOVER utility uses information in the SYSIBM.SYSCOPY catalog table to:

- Restore the page set with the data in the most recent full image copy (appearing, in Figure 44 on page 377, at X'40000').
- For table spaces only, apply all the changes summarized in any later incremental image copies. (There are two in Figure 44: X'80020' and X'C0040'.)
- Apply all changes to the page set that are registered in the log, beginning at the log RBA of the most recent image copy. (In Figure 44, X'C0040', that address is also stored in the SYSIBM.SYSCOPY catalog table.)

If the log has been damaged or discarded, or if data has been changed erroneously and then committed, you can recover to a particular point in time by limiting the range of log records to be applied by the RECOVER utility.

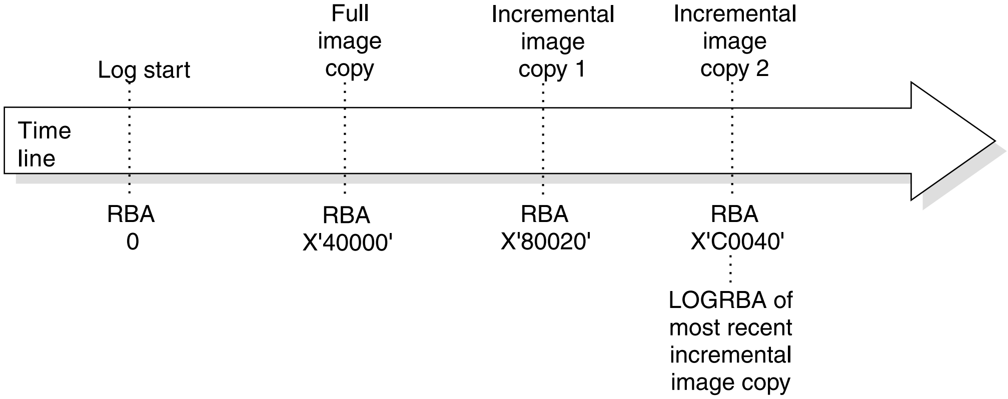


*Figure 44. Overview of DB2 recovery. The figure shows one complete cycle of image copies; the SYSIBM.SYSCOPY catalog table can record many complete cycles.*

## Complete recovery cycles

In planning for recovery, you determine how often to take image copies and how many complete cycles to keep. Those values tell how long you must keep log records and image copies for database recovery.

In deciding how often to take image copies, consider the time needed to recover a table space. It is determined by all of the following:

- The amount of log to traverse
- The time it takes an operator to mount and remove archive tape volumes
- The time it takes to read the part of the log needed for recovery
- The time needed to reprocess changed pages

In general, the more often you make image copies, the less time recovery takes; but, of course, the more time is spent making copies. If you use LOG NO without the COPYDDN keyword when you run the LOAD or REORG utilities, DB2 places the table space in copy pending status. You must remove the copy pending status of the table space by making an image copy before making further changes to the data. However, if you run REORG or LOAD REPLACE with the COPYDDN keyword, DB2 creates a full image copy of a table space during execution of the utility, so DB2 does not place the table space in copy pending status. Inline copies of indexes during LOAD and REORG are not supported.

If you use LOG YES and log all updates for table spaces, then an image copy of the table space is not required for data integrity. However, taking an image copy makes the recovery process more efficient. The process is even more efficient if you use MERGECOPY to merge incremental image copies with the latest full image copy. You can schedule the MERGECOPY operation at your own convenience, whereas the need for a recovery can come upon you unexpectedly. The MERGECOPY operation does not apply to indexes.

**Recommendation:** Copy your indexes after the associated utility has run. Indexes are placed in informational copy pending (ICOPY) status after running LOAD TABLESPACE, REORG TABLESPACE, REBUILD INDEX, or REORG INDEX utilities. Only structural modifications of the index are logged when these utilities are run, so there is not enough information in the log to recover the index.

Use the CHANGELIMIT option of the COPY utility to let DB2 determine when an image copy should be performed on a table space and whether a full or incremental copy should be taken. Use the CHANGELIMIT and REPORTONLY options together to let DB2 recommend what types of image copies to make. When you specify both CHANGELIMIT and REPORTONLY, DB2 makes no image copies. The CHANGELIMIT option does not apply to indexes.

In determining how many complete copy and log cycles to keep, you are guarding against damage to a volume containing an important image copy or a log data set. A retention period of at least two full cycles is recommended. For further security, keep records for three or more copy cycles.

## A recovery cycle example

Table 67 suggests how often a user group with 10 locally defined table spaces (one table per table space) might take image copies, based on frequency of updating. Their least-frequently-copied table is EMPSALS, containing employee salary data. If they choose to keep two complete image copy cycles on hand, then each time they copy EMPSALS they can delete records prior to its previous copy or copies, made two months ago. They will always have on hand between two months and four months of log records.

In the example, the user’s most critical tables are copied daily. Hence, the DB2 catalog and directory are also copied daily.

*Table 67. DB2 log management example*

| Table space name | Content | Update activity | Full image copy period |
|---|---|---|---|
| ORDERINF | Invoice line: part and quantity ordered | Heavy | Daily |
| SALESINF | Invoice description | Heavy | Daily |
| SALESQTA | Quota information for each sales person | Moderate | Weekly |
| SALESDSC | Customer descriptions | Moderate | Weekly |
| PARTSINV | Parts inventory | Moderate | Weekly |
| PARTSINF | Parts suppliers | Light | Monthly |
| PARTS | Parts descriptions | Light | Monthly |
| SALESCOM | Commission rates | Light | Monthly |
| EMPLOYEE | Employee descriptive data | Light | Monthly |
| EMPSALS | Employee salaries | Light | Bimonthly |

If you do a full recovery, you do not need to recover the indexes unless they are damaged. If you recover to a prior point in time, then you do need to recover the indexes. See “Considerations for recovering indexes” on page 375 for information on indexes.

## How DFSMShsm affects your recovery environment

The Data Facility Hierarchical Storage Manager (DFSMShsm) can automatically manage space and data availability among storage devices in your system. If you use it, you need to know that it automatically moves data to and from the DB2 databases.

DFSMShsm manages your disk space efficiently by moving data sets that have not been used recently to less expensive storage. It also makes your data available for recovery by automatically copying new or changed data sets to tape or disk. It can delete data sets, or move them to another device. Its operations occur daily, at a specified time, and allow for keeping a data set for a predetermined period before deleting or moving it.

All DFSMShsm operations can also be performed manually. *DFSMS/MVS: DFSMShsm Managing Your Own Data* tells how to use the DFSMShsm commands.

DFSMShsm:
- Uses cataloged data sets
- Operates on user tables, image copies, and logs
- Supports VSAM data sets

If a volume has a DB2 storage group specified, the volume should only be recalled to like devices of the same VOLSER defined by CREATE or ALTER STOGROUP.

DB2 can recall user page sets that have been migrated. Whether DFSMShsm recall occurs automatically is determined by the values of the RECALL DATABASE and RECALL DELAY fields of installation panel DSNTIPO. If the value of the RECALL DATABASE field is NO, automatic recall is not performed and the page set is considered an unavailable resource. It must be recalled explicitly before it can be used by DB2. If the value of the RECALL DATABASE field is YES, DFSMShsm is invoked to recall the page sets automatically. The program waits for the recall for the amount of time specified by the RECALL DELAY parameter. If the recall is not completed within that time, the program receives an error message indicating the page set is unavailable but that recall was initiated.

The deletion of DFSMShsm migrated data sets and the DB2 log retention period must be coordinated with use of the MODIFY utility. If not, you could need recovery image copies or logs that have been deleted. See “Discarding archive log records” on page 343 for suggestions.

## Making backup and recovery plans that maximize availability

You need to develop a plan for backup and recovery, and you need to become familiar enough with that plan so that when an outage occurs, you can get back in operation as quickly as possible. This topic contains some factors to consider when you develop and implement your plan.

***Decide on the level of availability you need:*** To do this, start by determining the primary types of outages you are likely to experience. Then, for each of those types of outages, decide on the maximum amount of time that you can spend on recovery. Consider the trade-off between cost and availability. Recovery plans for continuous availability are very costly, so you need to think about what percentage of the time your systems really need to be available.

***Practice for recovery:*** You cannot know whether a backup and recovery plan is workable unless you practice it. In addition, the pressure of a recovery situation can cause mistakes. The best way to minimize mistakes is to practice your recovery scenario until you know it well. The best time to practice is outside of regular working hours, when fewer key applications are running.

***Minimize preventable outages:*** One aspect of your backup and recovery plan should be eliminating the need to recover whenever possible. One way to do that is to prevent outages caused by errors in DB2. Be sure to check available maintenance often and apply fixes for problems that are likely to cause outages.

***Determine the required backup frequency:*** Use your recovery criteria to decide how often to make copies of your databases. For example, if the maximum acceptable recovery time after you lose a volume of data is two hours, your volumes typically hold about 4 GB of data, and you can read about 2 GB of data per hour, then you should make copies after every 4 GB of data written. You can use the COPY option SHRLEVEL CHANGE or DFSMSdss concurrent copy to make copies while transactions and batch jobs are running. You should also make a copy after running jobs that make large numbers of changes. In addition to copying your table spaces, you should also consider copying your indexes.

You can make additional backup image copies from a primary image copy by using the COPYTOCOPY utility. This capability is especially useful when the backup image is copied to a remote site that is to be used as a disaster recovery site for the local site. Applications can run concurrently with the COPYTOCOPY utility. Only utilities that write to the SYSCOPY catalog table cannot run concurrently with COPYTOCOPY.

***Minimize the elapsed time of RECOVER jobs:*** The RECOVER utility supports the recovery of a list of objects in parallel. For those objects in the list that can be processed independently, multiple subtasks are created to restore the image copies for the objects. The image copies must be on disk for the parallel function to be available. If an object that is on tape is encountered in the list, then processing for the remainder of the list waits until the processing of the tape object has completed.

***Minimize the elapsed time for copy jobs:*** You can use the COPY utility to make image copies of a list of objects in parallel. To take advantage of parallelism, image copies must be made to disk.

***Determine the right characteristics for your logs:***

- If you have enough disk space, use more and larger active logs. Recovery from active logs is quicker than from archive logs.
- To speed recovery from archive logs, consider archiving to disk.
- If you archive to tape, be sure you have enough tape drives that DB2 does not have to wait for an available drive on which to mount an archive tape during recovery.
- Make the buffer pools and the log buffers large enough to be efficient.

***Minimize DB2 restart time:*** Many recovery processes involve restart of DB2. You need to minimize the time that DB2 shutdown and startup take.

For non-data-sharing systems, you can limit the backout activity during DB2 system restart. You can postpone the backout of long running URs until after the DB2 system is operational. See “Deferring restart processing” on page 354 for an explanation of how to use the installation options LIMIT BACKOUT and BACKOUT DURATION to determine what backout work will be delayed during restart processing.

These are some major factors that influence the speed of DB2 shutdown:

- Number of open DB2 data sets

  During shutdown, DB2 must close and deallocate all data sets it uses if the fast shutdown feature has been disabled. The default is to use the fast shutdown feature. Contact your IBM service representative for information on enabling and disabling the fast shutdown feature. The maximum number of concurrently open data sets is determined by the DB2 subsystem parameter DSMAX. Closing and deallocation of data sets generally takes .1 to .3 seconds per data set. See Part 5 (Volume 2) of *DB2 Administration Guide* for information on how to choose an appropriate value for DSMAX.

Be aware that MVS global resource serialization (GRS) can increase the time to close DB2 data sets. If your DB2 data sets are not shared among more than one MVS system, set the GRS RESMIL parameter value to OFF or place the DB2 data sets in the SYSTEMS exclusion RNL. See *OS/390 MVS Planning: Global Resource Serialization* for details.

- Active threads

  DB2 cannot shut down until all threads have terminated. Issue the DB2 command -DISPLAY THREAD to determine if there are any active threads while DB2 is shutting down. If possible, cancel those threads.

- Processing of SMF data

  At DB2 shutdown, MVS does SMF processing for all DB2 data sets opened since DB2 startup. You can reduce the time that this processing takes by setting the MVS parameter DDCONS(NO).

These major factors influence the speed of DB2 startup:

- DB2 checkpoint interval

  The DB2 checkpoint interval creates a number of log records that DB2 writes between successive checkpoints. This value is controlled by the DB2 subsystem parameter CHKFREQ. The default of 50000 results in the fastest DB2 startup time in most cases.

  You can use the LOGLOAD or CHKTIME option of the SET LOG command to modify the CHKFREQ value dynamically without recycling DB2. The value you specify depends on your restart requirements. See “Changing the checkpoint frequency dynamically” on page 340 for examples of how you might use these command options. See Chapter 2 of *DB2 Command Reference* for detailed information about the SET LOG command.

- Long running units of work

  DB2 rolls back uncommitted work during startup. The amount of time for this activity is roughly double the time that the unit of work was running before DB2 shut down. For example, if a unit of work runs for two hours before a DB2 abend, it will take at least four hours to restart DB2. Decide how long you can afford for startup, and avoid units of work that run for more than half that long.

  You can use accounting traces to detect long running units of work. For tasks that modify tables, divide the elapsed time by the number of commit operations to get the average time between commit operations. Add commit operations to applications for which this time is unacceptable.

  **Recommendation**

  To detect long running units of recovery, enable the UR CHECK FREQ option of installation panel DSNTIPN. If long running units of recovery are unavoidable, consider enabling the LIMIT BACKOUT option on installation panel DSNTIPN.

- Size of active logs

  If you archive to tape, you can avoid unnecessary startup delays by making each active log big enough to hold the log records for a typical unit of work. This lessens the probability that DB2 will have to wait for tape mounts during startup. See Part 5 (Volume 2) of *DB2 Administration Guide* for more information on choosing the size of the active logs.

## How to find recovery information

This section contains guidance on locating and reporting information needed for recovery.

### Where recovery information resides

Information needed for recovery is contained in these locations:

- SYSIBM.SYSCOPY, a catalog table, contains information about full and incremental image copies. If concurrent updates were allowed when making the copy, the log RBA corresponds to the image copy start time; otherwise, it corresponds to the end time. The RECOVER utility uses the log RBA to look for log information after restoring the image copy. The SYSCOPY catalog table also contains information recorded by the COPYTOCOPY utility.

  SYSCOPY also contains entries with the same kinds of log RBAs recorded by the utilities QUIESCE, REORG, LOAD, REBUILD INDEX, RECOVER TOCOPY, and RECOVER TORBA (or TOLOGPOINT). For a summary of the information contained in the DB2 catalog tables, see Appendix D of *DB2 SQL Reference*.

  When the REORG utility is used, the time at which DB2 writes the log RBA to SYSIBM.SYSCOPY depends on the value of the SHRLEVEL parameter:

  – For SHRLEVEL NONE, the log RBA is written at the end of the reload phase.

    If a failure occurs before the end of the reload phase, the RBA is not written to SYSCOPY.

    If a failure occurs after the reload phase is complete (and thus, after the log RBA is written to SYSCOPY), the RBA is not backed out of SYSCOPY.

  – For SHRLEVEL REFERENCE and SHRLEVEL CHANGE, the log RBA is written at the end of the switch phase.

    If a failure occurs before the end of the switch phase, the RBA is not written to SYSCOPY.

    If a failure occurs after the switch phase is complete (and thus, after the log RBA is written to SYSCOPY), the RBA is not backed out of SYSCOPY.

  The log RBA is put in SYSCOPY whether the LOG option is YES or NO, or whether the UNLOAD PAUSE option is indicated.

  When DSNDB01.DBD01, DSNDB01.SYSUTILX, and DSNDB06.SYSCOPY are quiesced or copied, a SYSCOPY record is created for each table space and any associated index that has the COPY YES attribute. For recovery reasons, the SYSCOPY records for these three objects are placed in the log.

- SYSIBM.SYSLGRNX, a directory table, contains records of the log RBA ranges used during each period of time that any recoverable page set is open for update. Those records speed recovery by limiting the scan of the log for changes that must be applied.

  If you discard obsolete image copies, you should consider removing their records from SYSIBM.SYSCOPY and the obsolete log ranges from SYSIBM.SYSLGRNX. “Discarding SYSCOPY and SYSLGRNX records” on page 407 describes the process.

### Reporting recovery information

You can use the REPORT utility in planning for recovery. REPORT provides information necessary for recovering a page set. REPORT displays:

- Recovery information from the SYSIBM.SYSCOPY catalog table
- Log ranges of the table space from the SYSIBM.SYSLGRNX directory
- Archive log data sets from the bootstrap data set
- The names of all members of a table space set

You can also use REPORT to obtain recovery information about the catalog and directory.

Details about the REPORT utility and examples showing the results obtained when using the RECOVERY option are contained in Part 2 of *DB2 Utility Guide and Reference* .

## Preparing to recover to a prior point of consistency

The major steps in preparing to recover to a particular point in time are:

1. Release the data from any exception status.
2. Copy the data, taking appropriate precautions about concurrent activity.
3. Immediately after, establish a point when the data is consistent and no unit of work is changing it.

With that preparation, recovery to the point of consistency is as quick and simple as possible. DB2 begins recovery with the copy you made and reads the log only up to the point of consistency. At that point, there are no indoubt units of recovery to hinder restarting.

### Step 1: Resetting exception status

You can use the DISPLAY DATABASE RESTRICT command to determine whether the data is in an exception status. See Appendix C of *DB2 Utility Guide and Reference* for instructions on resetting those states.

### Step 2: Copying the data

You can copy the data and also establish a point of consistency for a list of objects, in one operation, by using the COPY utility with the option SHRLEVEL REFERENCE. That operation allows only read access to the data while it is copied. The data is consistent at the moment when copying starts and remains consistent until copying ends. The advantage of the method is that the data can be restarted at a point of consistency by restoring the copy only, with no need to read log records. The disadvantage is that updates cannot be made throughout the entire time that the data is being copied.

You can use the CONCURRENT option of the COPY utility to make a backup, with DFSMSdss concurrent copy, that is recorded in the DB2 catalog. For more information about using this option, see *DB2 Utility Guide and Reference*.

**Copying data while updates occur is not recommended.** However, to allow updates while the data is being copied, you can:

- Use the COPY utility with the option SHRLEVEL CHANGE.
- Use an offline program to copy the data, such as DSN1COPY, DFSMShsm, or disk dump.

**If you allow updates while copying**, then step 3 is essential. With concurrent updates, the copy can include uncommitted changes. Those might be backed out after copying ends. Thus, the copy is not necessarily consistent data, and recovery cannot rely on the copy only. Recovery requires reading the log up to a point of consistency, so you want to establish such a point as soon as possible.

### Step 3: Establishing a point of consistency

Use the QUIESCE utility also to establish a single point of consistency (a *quiesce point*) for one or more page sets. Typically, you name all the table spaces in a table space set that you want recovered to the same point in time to avoid referential integrity violations. Or you can use the QUIESCE utility with the TABLESPACESET keyword for RI-related tables. The following statement quiesces two table spaces in database DSN8D71A:

```
QUIESCE TABLESPACE DSN8D71A.DSN8S71E
        TABLESPACE DSN8D71A.DSN8S71D
```

QUIESCE writes changed pages from the page set to disk. The catalog table SYSIBM.SYSCOPY records the current RBA and the timestamp of the quiesce point. At that point, neither page set contains any uncommitted data. A row with ICTYPE Q is inserted into SYSCOPY for each table space quiesced. Page sets DSNDB06.SYSCOPY, DSNDB01.DBD01, and DSNDB01.SYSUTILX, are an exception: their information is written to the log. Indexes are quiesced automatically when you specify WRITE(YES) on the QUIESCE statement. A SYSIBM.SYSCOPY row with ICTYPE Q is inserted for indexes that have the COPY YES attribute.

QUIESCE allows concurrency with many other utilities; however, it does not allow concurrent updates until it has quiesced all specified page sets. Depending upon the amount of activity, that can take considerable time. Try to run QUIESCE when system activity is low.

Also, consider using the MODE(QUIESCE) option of the ARCHIVE LOG command when planning for offsite recovery. It creates a system-wide point of consistency, which can minimize the number of data inconsistencies when the archive log is used with the most current image copy during recovery. See “Archiving the log” on page 337 for more information about using the MODE(QUIESCE) option of the ARCHIVE LOG command.

## Preparing to recover the entire DB2 subsystem to a prior point in time

Under certain circumstances, you might want to reset the entire DB2 subsystem to a point of consistency. You can prepare a point of consistency by using the following procedure:

1. Display and resolve any indoubt units of recovery.
2. Use COPY to make image copies of all data, both user data and DB2 catalog and directory table spaces and optionally indexes. Copy SYSLGRNX and SYSCOPY last. Install job DSNTIJIC creates image copies of the DB2 catalog and directory table spaces. If you decide to copy your directory and catalog indexes, modify job DSNTIJIC to include those indexes. See Part 2 of *DB2 Installation Guide* for a description of job DSNTIJIC.

   Alternatively, you can use an offline method to copy the data. In that case, stop DB2 first; that is, do step 3 before step 2. If you do not stop DB2 before copying, you might have trouble restarting after restoring the system. If you do a volume restore, verify that the restored data is cataloged in the integrated catalog facility catalog. Use the access method services LISTCAT command to get a listing of the integrated catalog.
3. Stop DB2 by the command -STOP DB2 MODE (QUIESCE). DB2 does not actually stop until all currently executing programs have completed processing. Be sure to use MODE (QUIESCE); otherwise, I/O errors can occur when the steps listed in “Performing the fall back to a prior shutdown point” on page 495 are used to restart DB2.

4. When DB2 has stopped, use access method services EXPORT to copy all BSDS and active log data sets. If you have dual BSDSs or dual active log data sets, export both copies of the BSDS and the logs.
5. Save all the data that has been copied or dumped, and protect it and the archive log data sets from damage.

## Preparing for disaster recovery

In the case of a total loss of a DB2 computing center, you can recover on another DB2 system at a recovery site. To do this, you must regularly back up the data sets and the log for recovery. As with all data recovery operations, the objectives of disaster recovery are to lose as little data, workload processing (updates), and time as possible.

You can provide shorter restart times after system failures by using the installation options LIMIT BACKOUT and BACKOUT DURATION. These options postpone the backout processing of long running URs during DB2 restart. See Part 2 of *DB2 Installation Guide* for the details on how to use these parameters.

**Data sharing**

In a data sharing environment, you can use the LIGHT(YES) parameter to quickly bring up a DB2 member to recover retained locks. Restart light is not recommended for a restart in place and is intended only for a cross-system restart for a system that does not have adequate capacity to sustain the DB2 IRLM pair. Restart light can be used for normal restart and recovery. See Chapter 5 of *DB2 Data Sharing: Planning and Administration* for more details.

For data sharing, you need to consider whether you want the DB2 group to use light mode at the recovery site. A light start might be desirable if you have configured only minimal resources at the remote site. If this is the case, you might run a subset of the members *permanently* at the remote site. The other members are restarted and then directly shutdown. The procedure for a light start at the remote site is:

1. Start the members that run *permanently* with the LIGHT(NO) option. This is the default.
2. Start other members with LIGHT(YES). The members started with LIGHT(YES) use a smaller storage footprint. After their restart processing completes, they automatically shutdown. If ARM is in use, ARM does not automatically restart the members with LIGHT(YES) again.
3. Members started with LIGHT(NO) remain active and are available to run new work.

To keep ECSA storage consumption to a minimum, DB2 autostarts IRLM with PC = YES when restart light is invoked.

There are several levels of preparation for disaster recovery:

- Prepare the recovery site to recover to a fixed point in time.

  For example, you could copy everything weekly with a DFSMSdss volume dump (logical) and manually send it to the recovery site, then restore the data there.
- For recovery through the last archive, copy and send the following objects to the recovery site as you produce them:
  - Image copies of all catalog, directory, and user page sets
  - Archive logs

  - Integrated catalog facility catalog EXPORT and list
  - BSDS lists

    With this approach you can determine how often you want to make copies of essential recovery elements and send them to the recovery site.

  Once you establish your copy procedure and have it operating, you must prepare to recover your data at the recovery site. See “Remote site recovery from disaster at a local site” on page 449 for step-by-step instructions on the disaster recovery process.

- Use the log capture exit to capture log data in real time and send it to the recovery site. See “Reading log records with the log capture exit” on page 980 and “Log capture routines” on page 944.

## System-wide points of consistency

In any disaster recovery scenario, system-wide points of consistency are necessary for maintaining data integrity and preventing a loss of data. There is a direct relationship between the frequency with which you make and send copies to the recovery site and the amount of data that you could potentially lose.

Figure 45 is an overview of the process of preparing to bring DB2 up at a recovery site.

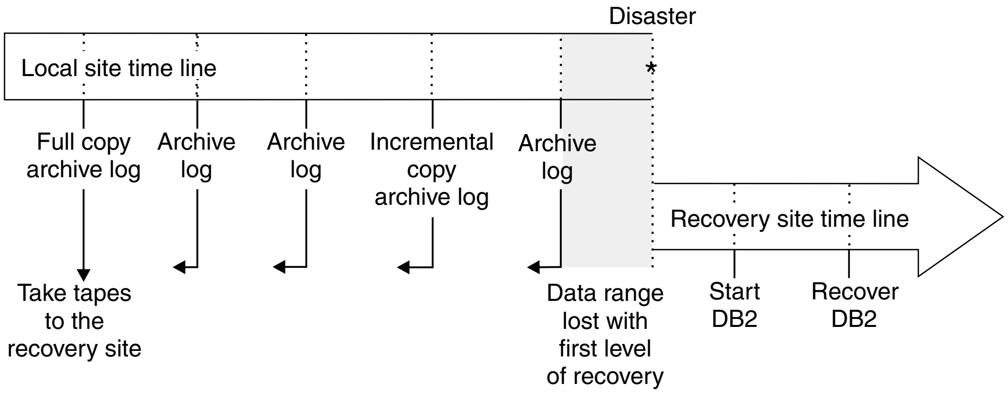


*Figure 45. Preparing for disaster recovery. The information you need to recover is contained in the copies of data (including the DB2 catalog and directory) and the archive log data sets.*

## Essential disaster recovery elements

Following is a list of essential disaster recovery elements and the steps you need to take to create them. You must determine how often to make copies and send them to the recovery site.

- Image copies
  1. Make copies of your data sets and DB2 catalogs and directories.

     Use the COPY utility to make copies for the local subsystem and additional | copies for disaster recovery. You can also use the COPYTOCOPY utility to | make additional image copies from the primary image copy made by the | COPY utility. Install your local subsystem with the LOCALSITE option of the SITE TYPE field on installation panel DSNTIPO. Use the RECOVERYDDN option when you run COPY to make additional copies for disaster recovery. You can use those copies on any DB2 subsystem that you have installed using the RECOVERYSITE option.[8]

   For information about making multiple image copies, see COPY and COPYTOCOPY in Part 2 of *DB2 Utility Guide and Reference*. Do not produce the copies by invoking COPY twice.

2. Catalog the image copies if you want to track them.
3. Create a QMF report or use SPUFI to issue a SELECT statement to list the contents of SYSCOPY.
4. Send the image copies and report to the recovery site.
5. Record this activity at the recovery site when the image copies and the report are received.

   All table spaces should have valid image copies. Indexes can have valid image copies or they can be rebuilt from the table spaces.

- Archive logs
  1. Make copies of the archive logs for the recovery site.
     a. Use the ARCHIVE LOG command to archive all current DB2 active log data sets. For more ARCHIVE LOG command information see “Archiving the log” on page 337.

        ***Recommendation:*** When using dual logging, keep both copies of the archive log at the local site in case the first copy becomes unreadable. If the first copy is unreadable, DB2 requests the second copy. If the second copy is not available, the read fails.

        However, if you take precautions when using dual logging, such as making another copy of the first archive log, you can send the second copy to the recovery site. If recovery is necessary at the recovery site, specify YES for the READ COPY2 ARCHIVE field on installation panel DSNTIPO. Using this option causes DB2 to request the second archive log first.

     b. Catalog the archive logs if you want to track them.

        You will probably need some way to track the volume serial numbers and data set names. One way of doing this is to catalog the archive logs to create a record of the necessary information. You could also create your own tracking method and do it manually.
  2. Use the print log map utility to create a BSDS report.
  3. Send the archive copy, the BSDS report, and any additional information about the archive log to the recovery site.
  4. Record this activity at the recovery site when the archive copy and the report are received.
- Integrated catalog facility catalog backups
  1. Back up all DB2-related integrated catalog facility catalogs with the VSAM EXPORT command on a daily basis.
  2. Synchronize the backups with the cataloging of image copies and archives.
  3. Use the VSAM LISTCAT command to create a list of the DB2 entries.
  4. Send the EXPORT backup and list to the recovery site.

[8] You can also use these copies on a subsystem installed with the LOCALSITE option if you run RECOVER with the RECOVERYSITE option. Or you can use copies prepared for the local site on a recovery site, if you run RECOVER with the option LOCALSITE.

5. Record this activity at the recovery site when the EXPORT backup and list are received.

- DB2 libraries
  1. Back up DB2 libraries to tape when they are changed. Include the SMP/E, load, distribution, and target libraries, as well as the most recent user applications and DBRMs.
  2. Back up the DSNTIJUZ job that builds the ZPARM and DECP modules.
  3. Back up the data set allocations for the BSDS, logs, directory, and catalogs.
  4. Document your backups.
  5. Send backups and corresponding documentation to the recovery site.
  6. Record activity at the recovery site when the library backup and documentation are received.

For disaster recovery to be successful, all copies and reports must be updated and sent to the recovery site regularly. Data will be up to date through the last archive sent. For disaster recovery start up procedures, see “Remote site recovery from disaster at a local site” on page 449.

## Ensuring more effective recovery from inconsistency problems

The DB2 RECOVER utility is often the quickest and easiest method of resolving data inconsistency problems. However, these problems can involve data that the RECOVER utility needs to use, such as the recovery log or image copy data sets. If the data needed by the RECOVER utility is damaged or unavailable, you might have to resolve the problem manually.

### Actions to take

To aid in successful recovery of inconsistent data:

- **During the installation of, or migration to, Version 7, make a full image copy of the DB2 directory and catalog using installation job DSNTIJIC.**

  See Part 2 of *DB2 Installation Guide* for DSNTIJIC information. If you did not do this during installation or migration, use the COPY utility, described in Part 2 of *DB2 Utility Guide and Reference*, to make a full image copy of the DB2 catalog and directory. If you do not do this and you subsequently have a problem with inconsistent data in the DB2 catalog or directory, you will not be able to use the RECOVER utility to resolve the problem.

- **Periodically make an image copy of the catalog, directory, and user databases.**

  This minimizes the time the RECOVER utility requires to perform recovery. In addition, this increases the probability that the necessary archive log data sets will still be available. You should keep *two* copies of each level of image copy data set. This reduces the risk involved if one image copy data set is lost or damaged. See Part 2 of *DB2 Utility Guide and Reference* for more information about using the COPY utility.

- **Use dual logging for your active log, archive log, and bootstrap data sets.**

  This increases the probability that you can recover from unexpected problems. It is especially useful in resolving data inconsistency problems. See “Establishing the logging environment” on page 333 for related dual logging information.

- **Before using RECOVER, rename your data sets.**

  If the image copy or log data sets are damaged, you can compound your problem by using the RECOVER utility. Therefore, before using RECOVER, rename your data sets by using one of the following methods:

  – rename the data sets that contain the page sets you want to recover, or

  - copy your data sets using DSN1COPY, or
  - for user-defined data sets, use access method services to define a new data set with the original name.

  The RECOVER utility applies log records to the new data set with the old name. Then, if a problem occurs during RECOVER utility processing, you will have a copy (under a different name) of the data set you want to recover.

- **Keep back-level image copy data sets.**

  If you make an image copy of a page set containing inconsistent data, the RECOVER utility cannot resolve the data inconsistency problem. However, you can use RECOVER TOCOPY or TOLOGPOINT to resolve the inconsistency if you have an older image copy of the page set that was taken before the problem occurred. You can also resolve the inconsistency problem by using a point-in-time recovery to avoid using the most recent image copy.

- **Maintain consistency between related objects.**

  A referential structure is a set of tables including indexes and their relationships. It must include at least one table, and for every table in the set, include all of the relationships in which the table participates, as well as all the tables to which it is related. To help maintain referential consistency, keep the number of table spaces in a table space set to a minimum, and avoid tables of different referential structures in the same table space. The TABLESPACESET option of the REPORT utility reports all members of a table space set defined by referential constraints.

  A referential structure must be kept consistent with respect to point-in-time recovery. Use the QUIESCE utility to establish a point of consistency for a table space set, to which the table space set can later be recovered without introducing referential constraint violations.

  A base table space must be kept consistent with its associated LOB table spaces with respect to point-in-time recovery. Use the TABLESPACESET option of the REPORT utility to find all LOB table spaces associated with a base table space. Use the QUIESCE utility to establish a point of consistency, for a table space set, to which the table space set can later be recovered.

## Actions to avoid

- **Do not discard archive logs you might need.**

  The RECOVER utility might need an archive log to recover from an inconsistent data problem. If you have discarded it, you cannot use the RECOVER utility and must resolve the problem manually. For information about determining when you can discard archive logs, see “Discarding archive log records” on page 343.

- **Do not make an image copy of a page set that contains inconsistent data.**

  If you use the COPY utility to make an image copy of a page set containing inconsistent data, the RECOVER cannot recover a problem involving that page set unless you have an older image copy of that page set taken before the problem occurred. You can run DSN1COPY with the CHECK option to determine whether intra-page data inconsistency problems exist on page sets before making image copies of them. If you are taking a copy of a catalog or directory page set, you can run DSN1CHKR which verifies the integrity of the links, and the CHECK DATA utility which checks the DB2 catalog (DSNDB06). For information, see *DB2 Utility Guide and Reference*.

- **Do not use the TERM UTILITY command on utility jobs you want to restart.**

  If an error occurs while a utility is running, the data on which the utility was operating might continue to be written beyond the commit point. If the utility is restarted later, processing resumes at the commit point or at the beginning of the current phase, depending on the restart parameter that was specified. If the utility stops while it has exclusive access to data, other applications cannot access that data. In this case, you might want to issue the TERM UTILITY command to terminate the utility and make the data available to other applications. However, use the TERM UTILITY command only if you cannot restart or do not need to restart the utility job.

When you issue the TERM UTILITY command, two different situations can occur:

– If the utility is active, it terminates at its next commit point.

– If the utility is stopped, it terminates immediately.

If you use the TERM UTILITY command to terminate a utility, the objects on which the utility was operating are left in an indeterminate state. Often, the same utility job cannot be rerun. The specific considerations vary for each utility, depending on the phase in process when you issue the command. For details, see Part 2 of *DB2 Utility Guide and Reference*.

## Running RECOVER in parallel

You can schedule jobs with the RECOVER utility in two ways:

- Use the PARALLEL keyword on the RECOVER utility to support the recovery of a list of objects in parallel. For those objects in the list that can be processed independently, multiple subtasks are created to restore the image copies for the objects. The image copies must be on disk for the parallel function to be available; however, parallelism is not affected if the copy on disk has been migrated to tape by DFSMShsm. If an object that is on tape is encountered in the list, then processing for the remainder of the list will pause until the object being restored from tape completes.

  When you use one utility statement to recover indexes and table spaces, the logs for all indexes and tables spaces are processed in one pass. This approach results in a significant performance advantage, especially when the required archive log data is on tape or the fast log apply function is enabled, or both of these conditions occur.

- Schedule concurrent RECOVER jobs that process different partitions. The degree of parallelism in this case is limited by contention for both the image copies and the required log data.

  Image copies that reside on disk are read in parallel. Image copies that reside on tape are read serially by each RECOVER job in turn.

  Log data is read by concurrent jobs as follows:

  – Active logs and archive logs on disk are read entirely in parallel.

  – A data set controlled by DFSMShsm is first recalled. It then resides on disk and is read in parallel.

  – A non-DFSMShsm data set that must be read from tape is read sequentially by each job in turn. As soon as one job finishes with a tape, the next job gains control and begins reading the same tape. Different jobs can read different log tapes in parallel. DB2 optimizes the tape handling process.

## Using fast log apply during RECOVER

The RECOVER utility automatically uses the fast log apply process during the LOGAPPLY phase if fast log apply has been enabled on the DB2 subsystem. For detailed information about defining storage for the sort used in fast log apply processing, see the Log Apply Storage field on panel DSNTIPL in *DB2 Installation Guide*.

## Reading the log without RECOVER

The DATA CAPTURE(CHANGES) clause of the SQL statements CREATE TABLE and ALTER TABLE captures all SQL data changes made to the table on the DB2 log. The captured data can be propagated to an IMS subsystem or remain in the DB2 log. This allows the creation of duplicate data for recovery purposes. Although SQL changes to tables defined for data capture are supported from any subsystem, propagation is permitted only to an IMS subsystem. For further information see “Appendix C. Reading log records” on page 957.

Data written to the log for propagation to IMS uses an expanded format that is much longer than the DB2 internal format. Using DATA CAPTURE(CHANGES) can greatly increase the size of your log.

## Copying page sets and data sets

You can use the COPY utility to copy data from a page set to an MVS sequential data set on disk or tape. It makes a full or incremental image copy, as you choose, and it can be used to make copies that will be used for local or offsite recovery operations. Use the COPYTOCOPY utility to make additional image copies from a primary image copy that you made with the COPY utility.

A full image copy is required for indexes. For information about copying indexes, see “Considerations for recovering indexes” on page 375.

You can use the CONCURRENT option of the COPY utility to make a copy, with DFSMSdss concurrent copy, that is recorded in the DB2 catalog. For more information about using this option, see *DB2 Utility Guide and Reference*.

Use the MERGECOPY utility to merge several image copies. MERGECOPY does not apply to indexes.

The CHANGELIMIT option of the COPY utility causes DB2 to make an image copy automatically when a table space has changed past a default limit or a limit you specify. DB2 determines whether to make a full or incremental image copy. DB2 makes an incremental image copy if the percent of changed pages is greater than the low CHANGELIMIT value and less than the high CHANGELIMIT value. DB2 makes a full image copy if the percent of changed pages is greater than or equal to the high CHANGELIMIT value. The CHANGELIMIT option does not apply to indexes.

If you want DB2 to recommend what image copies should be made but not make the image copies, use the CHANGELIMIT and REPORTONLY options of the COPY utility.

If you specify the parameter DSNUM ALL with CHANGELIMIT and REPORTONLY, DB2 reports information for each partition of a partitioned table space or each piece of a nonpartitioned table space.

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. When you use the COPY utility with the CHANGELIMIT option to display image copy statistics, the COPY utility uses the following return codes to indicate the degree that a table space or list of table spaces has changed:

**Code Meaning**

**1** Successful and no CHANGELIMIT value is met. No image copy is recommended or taken.

**2** Successful and the percent of changed pages is greater than the low CHANGELIMIT value and less than the high CHANGELIMIT value. An incremental image copy is recommended or taken.

**3** Successful and the percent of changed pages is greater than or equal to the high CHANGELIMIT value. A full image copy is recommended or taken.

When you use generation data groups (GDGs) and need to make an incremental image copy, there are new steps you can take to prevent an empty image copy output data set from being created if no pages have been changed. You can do the following:

- Make a copy of your image copy step, but add the REPORTONLY and CHANGELIMIT options to the new COPY utility statement. The REPORTONLY keyword specifies that you only want image copy information displayed. Change the SYSCOPY DD card to DD DUMMY so that no output data set is allocated. Run this step to visually determine the change status of your table space.
- Add this step before your existing image copy step, and add a JCL conditional statement to examine the return code and execute the image copy step if the table space changes meet either of the CHANGELIMIT values.

You can also use the COPY utility with the CHANGELIMIT option to determine whether any space map pages are broken, or to identify any other problems that might prevent an image copy from being taken, such as the object being in recover pending status. You need to correct these problems before you run the image copy job.

You can also make a full image copy when you run the LOAD or REORG utility. This technique is better than running the COPY utility after the LOAD or REORG utility because it decreases the time that your table spaces are unavailable. However, only the COPY utility makes image copies of indexes.

***Related information:*** For guidance in using COPY and MERGECOPY and making image copies during LOAD and REORG, see Part 2 of *DB2 Utility Guide and Reference*.

***Backing up with DFSMS:*** The concurrent copy function of Data Facility Storage Management Subsystem (DFSMS) can copy a data set concurrently with access by other processes, without significant impact on application performance. The function requires the 3990 Model 3 controller with the extended platform.

There are two ways to use the concurrent copy function of Data Facility Storage Management Subsystem (DFSMS):

- Run the COPY utility with the CONCURRENT option. DB2 records the resulting image copies in SYSIBM.SYSCOPY. To recover with these DFSMS copies, you can run the RECOVER utility to restore those image copies and apply the necessary log records to them to complete recovery.
- Make copies using DFSMS outside of DB2's control. To recover with these copies, you must manually restore the data sets, and then run RECOVER with the LOGONLY option to apply the necessary log records.

***Backing up with RVA storage control or Enterprise Storage Server™ :*** IBM's RAMAC® Virtual Array (RVA) storage control with the peer-to-peer remote copy (PPRC) function or Enterprise Storage Server provides a faster method of recovering DB2 subsystems at a remote site in the event of a disaster at the local site. You can use RVAs, PPRC, and the RVA fast copy function, SnapShot, to create entire DB2 subsystem backups to a point-in-time on a *hot stand-by* remote site without interruption of any application process. Another option is to use the Enterprise Storage Server FlashCopy function to create point-in-time backups of entire DB2 subsystems.

To use RVA SnapShot or Enterprise Storage Server FlashCopy for a DB2 backup requires a method of suspending all update activity for a DB2 subsystem to make a remote copy of the entire subsystem without quiescing the update activity at the primary site. Use the SUSPEND option on the -SET LOG command to suspend all logging activity at the primary site which also prevents any database updates. After the remote copy has been created, use the RESUME option on the -SET LOG command to return to normal logging activities. See the *DB2 Command Reference* for more details on using the -SET LOG command. For more information about RVA, see *IBM RAMAC Virtual Array*. For more information on using PPRC, see *RAMAC Virtual Array: Implementing Peer-to-Peer Remote Copy*. For more information about Enterprise Storage Server and the FlashCopy function, see *Enterprise Storage Server Introduction and Planning*.

## Recovering page sets and data sets

You can recover objects in either of these ways:

- If you made backup copies of table spaces using the COPY utility, the COPYTOCOPY utility, or the inline copy feature of the LOAD or REORG utility, use the RECOVER utility to restore the objects to the current or a previous state. Backup copies of indexes are made using the DB2 COPY utility.
- If you made backup copies using a method outside of DB2’s control, such as with DSN1COPY or the DFSMSdss concurrent copy function, use the same method to restore the objects to a prior point in time. Then, if you wish to restore the objects to currency, run the RECOVER utility with the LOGONLY option.

The RECOVER utility performs these actions:

- Restores the most current full image copy
- Applies changes recorded in later incremental image copies of table spaces, if applicable, and applies later changes from the archive or active log

RECOVER can act on:

- A table space, or list of table spaces
- An index, or list of indexes
- A mixed list of table spaces and indexes
- A specific partition within an index space
- A single page
- A page range within a table space that DB2 has found in error
- The catalog and directory

Typically, RECOVER restores an object to its current state by applying all image copies and log records. It can also restore to a prior state, meaning one of the following:

- A specified point on the log (use the TORBA or TOLOGPOINT keyword)
- A particular image copy (use the TOCOPY keyword)

The RECOVER utility can use image copies for the local site or the recovery site, regardless of where you invoke the utility. The RECOVER utility locates all full and incremental image copies.

The RECOVER utility first attempts to use the primary image copy data set. If an error is encountered (allocation, open, or I/O), RECOVER attempts to use the backup image copy if it is present. If an error is encountered with the backup copy, RECOVER falls back to an earlier recoverable point.

For guidance in using RECOVER and REBUILD INDEX, see Part 2 of *DB2 Utility Guide and Reference* .

Not every recovery operation requires RECOVER; see also
“Recovering error ranges for a work file table space” on page 395
“Recovering the work file database”
“Recovering data to a prior point of consistency” on page 396.

***A caution about disk dump:*** Be very careful when using disk dump and restore for recovering a data set. Disk dump and restore can make one data set inconsistent with DB2 subsystem tables in some other data set. Use disk dump and restore only to restore the entire subsystem to a previous point of consistency, and prepare that point as described in the alternative in step 2 under “Preparing to recover to a prior point of consistency” on page 383.

## Recovering the work file database

You *cannot* use RECOVER with the work file database (called DSNDB07, except in a data sharing environment). That database is used for temporary space for certain SQL operations, such as join and ORDER BY. If DSNDB01.DBD01 is stopped or otherwise inaccessible when DB2 is started, then the descriptor for the work file database is not loaded into main storage and the work file database is not allocated. In order to recover from this condition after DSNDB01.DBD01 has been made available, it is necessary to stop and then start the work file database again.

### Problem with user-defined work file data sets

If you have a problem on a volume of a user-defined data set for the work file database, then:

1. Issue the following DB2 command:

   ```
   -STOP DATABASE (DSNDB07)
   ```

2. Use the DELETE and DEFINE functions of access method services to redefine a user work file on a different volume and reconnect it to DB2.
3. Issue the following DB2 command:

   ```
   -START DATABASE (DSNDB07)
   ```

### Problem with DB2-managed work file data sets

If you have a problem on a volume in a DB2 storage group for the work file database, such as a system I/O problem, then:

1. Enter the following SQL statement to remove the problem volume from the DB2 storage group:

   ```
   ALTER STOGROUP stogroup-name
    REMOVE VOLUMES (xxxxxx);
   ```

2. Issue the following DB2 command:

   ```
   -STOP DATABASE (DSNDB07)
   ```

3. Enter the following SQL statement to drop the table space with the problem:

   ```
   DROP TABLESPACE DSNDB07.tsname:
   ```

4. Re-create the table space. You can use the same storage group, because the problem volume has been removed, or you can use an alternate.

```
CREATE TABLESPACE tsname
  IN DSNDB07
  USING STOGROUP stogroup-name;
```

5. Issue the following command:

```
-START DATABASE (DSNDB07)
```

### Recovering error ranges for a work file table space

Page error ranges operate for work file table spaces in the same way as for other DB2 table spaces, except for the process of recovering them. Error ranges in a work file table space cannot be reset by RECOVER ERROR RANGE. Instead, do the following:

1. Stop the work file table space.
2. Correct the disk error, using the ICKDSF service utility or access method services to delete and redefine the data set.
3. Start the work file table space. When the work file table space is started, DB2 automatically resets the error range.

Also, DB2 always resets any error ranges when the work file table space is initialized, regardless of whether the disk error has really been corrected. Work file table spaces are initialized when:

- The work file table space is stopped and then started
- The work file database is stopped and then started, and the work file table space was not previously stopped
- DB2 is started and the work file table space was not previously stopped

If the error range is reset while the disk error still exists, and if DB2 has an I/O error when using the work file table space again, then DB2 sets the error range again.

## Recovering the catalog and directory

Catalog and directory objects must be recovered in a particular order. Because the recovery of some objects depends on information derived from others, recovery cannot proceed until the logically prior objects are in an undamaged state. For this reason, you cannot recover a catalog or directory page set as part of a list of page sets. You can recover a catalog or directory table space with its corresponding IBM-defined indexes in a list. The restriction of only one catalog or directory table space per list still applies. See the description of RECOVER in Part 2 of *DB2 Utility Guide and Reference* for more information about the specific order of recovery.

You can use the REPORT utility to report on recovery information about the catalog and directory.

To avoid restart processing of any page sets before attempts are made to recover any of the members of the list of catalog and directory objects, use the DEFER option when installing DB2 followed by the option ALL. For more information on DEFER, see “Deferring restart processing” on page 354.

***Point-in-time recovery:*** Recovering the DB2 catalog and directory to a prior point-in-time can cause a populated VSAM data set that was defined with DEFINE NO option to revert back to the undefined state. To avoid errors, you must delete the existing VSAM data sets before the table space or index can be accessed. For more information on the DEFINE NO option of the CREATE TABLESPACE and CREATE INDEX SQL statements, see *DB2 SQL Reference*.

A prior point-in-time recovery on the catalog and directory can also cause problems for user table spaces or index spaces that have been reorganized with FASTSWITCH. If the IPREFIX recorded in the DB2 catalog and directory is different from the VSAM cluster names, you cannot access your data. To determine which IPREFIX is recorded in the catalog for a particular object, query the SYSIBM.SYSTABLEPART or SYSIBM.SYSINDEXPART catalog table. Then rename any VSAM clusters whose names do not specify the correct IPREFIX. For example, if the IPREFIX value in the catalog is J, the cluster name should be:

```
catname.DSNDBC.dbname.spname.J0001.A001
```

***Recovery after a conditional restart of DB2:*** After a DB2 conditional restart in which a log record range is specified, such as with a cold start, a portion of the DB2 recovery log is no longer available. If the unavailable portion includes information that is needed for internal DB2 processing, an attempt to use the RECOVER utility to restore directory table spaces DSNDBD01 or SYSUTILX, or catalog table space SYSCOPY, will fail with abend 00E40119. Instead of using the RECOVER utility, use this procedure to recover those table spaces and their indexes:

1. Run DSN1COPY to recover the table spaces from an image copy.
2. Run the RECOVER utility with the LOGONLY option to apply updates from the log records to the recovered table spaces.
3. Rebuild the indexes.
4. Make a full image copy of the table spaces and optionally the indexes to establish a new recovery point.

## Recovering data to a prior point of consistency

Data can be restored to its state at a prior point in time if it has been backed up appropriately. There are several ways to restore data, described in the following sections:

- “Restoring data by using DSN1COPY” on page 399
- “Backing up and restoring data with non-DB2 dump and restore” on page 400
- “Using RECOVER to restore data to a previous point in time” on page 400.

You cannot recover to certain points in time. See Part 2 of *DB2 Utility Guide and Reference* for more information about those restrictions.

The following considerations apply to all methods for backing up and restoring data:

***Considerations for recovering table space sets:*** To determine a valid quiesce point for the table space set, use the procedure for determining a RECOVER TORBA value. See RECOVER in Part 2 of *DB2 Utility Guide and Reference* for more information.

***Be aware of table space sets:*** If you restore a page set to a prior state, restore all related tables and indexes to the same point to avoid inconsistencies. The table spaces that contain related tables are called a *table space set*; similarly, a LOB table space and its associated base table space are also part of a table space set. For example, in the DB2 sample application, a column in the EMPLOYEE table identifies the department to which each employee belongs. The departments are described by records in the DEPARTMENT table, which is in a different table space. If only that table space is restored to a prior state, a row in the unrestored EMPLOYEE table might then identify a department that does not exist in the restored DEPARTMENT table.

You can use the REPORT utility to determine all the page sets that belong to a single table space set and then restore those page sets that are related. However, if there are related page sets that belong to more than one table space set or there are page sets that are logically related in application programs of which DB2 is not aware, you are responsible for identifying all the page sets on your own.

***Recovering indexes:*** If image copies exist for the indexes, use the RECOVER utility. If indexes do not have image copies, use REBUILD INDEX to re-create the indexes after the data has been recovered.

***Recovering a table with identity columns:*** When data is recovered to a prior point-in-time on a table space that contains a table with an identity column, consider the following two cases:

- Assume the table was created with an identity column. If the table space is recovered to a prior point-in-time, the RECOVER utility does *not* set the REORP status for the table space, and the table is ready to access.

  The values for the identity columns of the rows that exist after recovery are the same values for these rows as before recovery. However, a large gap in the sequence of generated values for the identity column might result when the next row is inserted. For example, assume that a table has an identity column that increments by 1 and that the last generated value at time T1 was 100 and DB2 subsequently generates values up to 1000. Now, assume that the table space is recovered back to time T1. The generated value of the identity column for the next row inserted after the recovery completes will be 1001, leaving a gap from 101 to 1000 in the values of the identity column.
- Assume an identity column was added to the table after the table was created. When the column was added, the REORG utility was run to reset the REORP status, and DB2 generated the values for the identity columns in all existing rows. Now, if the table space is recovered to a point-in-time prior to when the identity column was added to the table, the RECOVER utility sets the table space status to REORP. The RECOVER utility also sets the check pending status if the table is a member of a referential set.

  To remove the various pending states, run the following utilities in this order:

  1. Use the REORG utility to remove the REORP status.

     When REORG assigns the identity column values, it does so based on the *current* ordinal position of the rows in the table, beginning with the start value. As a result, if the number of rows in the table after recovery is different than the original number of rows when the table was first altered, the identity column values after this REORG might be different than the original identity column values.
  2. If the table space status is auxiliary check pending:
     - Use CHECK LOB for all associated LOB table spaces.
     - Use CHECK INDEX for all indexes on the LOB table spaces.
  3. Use the CHECK DATA utility to remove the check-pending status.

  For the ADD COLUMN case, if the table space is partitioned, all partitions are marked REORP after a point-in-time recovery, and all partitions must be recovered.

***Check consistency with catalog definitions:*** Catalog and data inconsistencies are usually the result of one of the following:

- A catalog table space was restored.

- If SYSSEQ and SYSSEQ2 are recovered to a prior point-in-time, DB2 might generate some duplicate values for some identity columns. To avoid any duplicate values, table spaces that contain tables with identity columns should be recovered to the same prior point-in-time.
- The definition of a table or table space changed after the data was last backed up.

If restoring your data might have caused an inconsistency between your catalog and data, you need to do the following:

1. Run the DSN1PRNT utility with the FORMAT option against all data sets that might contain user table spaces. These data sets are of the form

   ```
   catname.DSNDBC.dbname.tsname.y0001.A00n
   ```

   where *y* can be either I or J.
2. Execute these SELECT statements to find a list of table space and table definitions in the DB2 catalog:

   **Product-sensitive Programming Interface**

   ```
   SELECT NAME, DBID, PSID FROM SYSIBM.SYSTABLESPACE;
   SELECT NAME, TSNAME, DBID, OBID FROM SYSIBM.SYSTABLES;
   ```

   **End of Product-sensitive Programming Interface**
3. For each table space name in the catalog, check to see if there is a data set with a corresponding name. If a data set exists,
   - Find the field HGBOBID in the header page section of the DSN1PRNT output. This field contains the DBID and PSID for the table space. See if the corresponding table space name in the DB2 catalog has the same DBID and PSID.
   - If the DBID and PSID do not match, execute DROP TABLESPACE and CREATE TABLESPACE to replace the incorrect table space entry in the DB2 catalog with a new entry. Be sure to make the new table space definition exactly like the old one. If the table space is segmented, SEGSIZE must be identical for the old and new definitions.

     A LOB table space can be dropped only if it is empty (that is, it does not contain auxiliary tables). If a LOB table space is not empty, you must first drop the auxiliary table before you drop the LOB table space. To drop the auxiliary table, do one of the following actions:
     - Drop the base table, or
     - Delete all rows from the base table and then drop the auxiliary table, or
     - Update all LOBs in the LOB table space to null or zero-length string and then drop the auxiliary table.
   - Find the PGSOBD fields in the data page sections of the DSN1PRNT output. These fields contain the OBIDs for the tables in the table space. For each OBID you find in the DSN1PRNT output, search the DB2 catalog for a table definition with the same OBID.
   - If any of the OBIDs in the table space do not have matching table definitions, examine the DSN1PRNT output to determine the structure of the tables associated with these OBIDs. If a table exists whose structure matches a definition in the catalog, but the OBIDs differ, proceed to the next step. The OBIDXLAT option of DSN1COPY will correct the mismatch. If a table exists for which there is no table definition in the catalog, re-create the table definition using CREATE TABLE. To re-create a table definition for a table

that has had columns added, first use the **original** CREATE TABLE statement, then use ALTER TABLE to add columns to make the table definition match the current structure of the table.

- Use the utility DSN1COPY with the OBIDXLAT option to copy the existing data to the *new* tables and table space and translate the DBID, PSID, and OBIDs.

If a table space name in the DB2 catalog does not have a data set with a corresponding name, the table space was probably created after your backup was taken, and you cannot recover the table space. Execute DROP TABLESPACE to delete the entry from the DB2 catalog.

4. For each data set in the DSN1PRNT output, check to see if there is a corresponding DB2 catalog entry. If no entry exists, follow the instructions in "Recovery of an accidentally dropped table space" on page 405 to re-create the entry in the DB2 catalog.

See Part 3 of *DB2 Utility Guide and Reference* for more information about DSN1COPY and DSN1PRNT.

***Recovery of segmented table spaces:*** When data is restored to a prior point in time on a segmented table space, information in the DBD for the table space might not match the restored table space. If you use the DB2 RECOVER utility, the DBD is updated dynamically to match the restored table space on the next non-index access of the table. The table space must be in WRITE access mode. If you use a method outside of DB2's control, such as DSN1COPY, to restore the table space to a prior point in time, run the REPAIR utility with the LEVELID option to force DB2 to accept the down-level data, then run the REORG utility on the table space to correct the DBD.

***Catalog and directory:*** If any table space in the DB2 catalog (DSNDB06) and directory (DSNDB01) is recovered, then all table spaces (except SYSUTILX) must be recovered.

The catalog and directory contain definitions of all databases. When databases DSNDB01 and DSNDB06 are restored to a prior point, information about later definitions, authorizations, binds, and recoveries is lost. If you restore the catalog and directory, you might have to restore user databases; if you restore user databases, you might have to restore the catalog and directory.

## Restoring data by using DSN1COPY

You can use DSN1COPY to restore data that has been previously backed up by DSN1COPY or by COPY. If you use DSN1COPY to restore data or move data, the data definitions for the target object must be exactly the same as when the copy was created. You cannot use DSN1COPY to restore data that was backed up with the DFSMSdss concurrent copy facility.

Be careful when creating backups with DSN1COPY. You must ensure that the data is consistent or you will end up with faulty backup copies. One advantage of using COPY to create backups is that it does not allow you to copy data that is in check or recovery pending status. COPY allows you to prepare an up-to-date image copy of the table space, either by making a full image copy or by making an incremental image copy and merging it with the most recent full image copy.

Keep access method services LISTCAT listings of table space data sets that correspond to each level of retained backup data.

For more information about using DSN1COPY, see Part 3 of *DB2 Utility Guide and Reference*.

## Backing up and restoring data with non-DB2 dump and restore

You can use certain non-DB2 facilities to dump and restore data sets and volumes. But note carefully the limitations described below.

Even though DB2 data sets are defined as VSAM data sets, DB2 data cannot be read or written by VSAM record processing because it has a different internal format. The data can be accessed by VSAM control interval (CI) processing. If you manage your own data sets, you can define them as VSAM linear data sets (LDSs), and access them through services that support data sets of that type.

Access method services for CI and LDS processing are available in MVS. IMPORT and EXPORT use CI processing; PRINT and REPRO do not, but do support LDSs.

DFSMS/MVS Data Set Services (DFSMSdss) is available on MVS and provides dump and restore services that can be used on DB2 data sets. Those services do use VSAM CI processing.

## Using RECOVER to restore data to a previous point in time

TOCOPY, TORBA and TOLOGPOINT are options of the RECOVER utility. All terminate recovery at a specified point. Because they recover data to a prior time, and not to the present, they are referred to as *point-in-time* recoveries. A recovery to a prior point in time will use either the TOCOPY, TORBA, or TOLOGPOINT options of RECOVER.

TOCOPY identifies an image copy. Recovery is restored to the value of that copy, without applying subsequent changes from the log. If the image copy in TOCOPY cannot be applied, RECOVER TOCOPY uses an earlier full image copy and applies logged changes up to the specified point.

If the image copy data set is cataloged when the image copy is made, then the entry for that copy in SYSIBM.SYSCOPY does not record the volume serial numbers of the data set. Identify that copy by its name, using TOCOPY *data set name*. If the image copy data set was not cataloged when created, then you can identify the copy by its volume serial identifier, using TOVOLUME *volser*.

In a non-data-sharing environment, TORBA and TOLOGPOINT are interchangeable keywords that identify an RBA on the log at which recovery stops. TORBA can be used in a data sharing environment only if the TORBA value is before the point at which data sharing was enabled. In this publication, whenever we talk about using the TORBA keyword, the TOLOGPOINT keyword can be used instead. If you are planning to use data sharing eventually, start using TOLOGPOINT now, to prepare.

With TORBA and TOLOGPOINT, the most recent full image copy taken before that point on the log is restored, and logged changes are applied up to, and including, the record that contains the specified log point. If no full image copy exists before the chosen log point, recovery is attempted entirely from the log, applying the log from page set creation to the chosen log point. This assumes you have not used the MODIFY RECOVERY utility to delete SYSIBM.SYSLGRNX records for the page set.

If you are working with partitioned table spaces, image copies taken prior to resetting the REORG pending status of any partition of a partitioned table space cannot be used for recovery to currency. Avoid performing a point-in-time recovery for a partitioned table space to a point-in-time that is after the REORG pending status was set, but before a rebalancing REORG was performed. See information about RECOVER in Part 2 of *DB2 Utility Guide and Reference* for details on determining an appropriate point in time and creating a new recovery point.

***Planning for point-in-time recovery:*** TOCOPY and TORBA are viable alternatives in many situations in which recovery to the current point in time is not possible or desirable. To make these options work best for you, take periodic quiesce points at points of consistency that are appropriate to your applications.

When making copies of a single object, use SHRLEVEL REFERENCE to establish consistent points for TOCOPY recovery. Copies made with SHRLEVEL CHANGE do not copy data at a single instant, because changes can occur as the copy is made. A subsequent RECOVER TOCOPY operation can produce inconsistent data.

When copying a list of objects, use SHRLEVEL REFERENCE. If a subsequent recovery to a point-in-time is necessary, you can use a single RECOVER utility statement to list all of the objects, along with TOLOGPOINT to identify the common RBA or LRSN value. If you use SHRLEVEL CHANGE to copy a list of objects, you should follow it with a QUIESCE of the objects.

An inline copy made during LOAD REPLACE can produce unpredictable results if that copy is used later in a RECOVER TOCOPY operation. DB2 makes the copy during the RELOAD phase of the LOAD operation. Therefore, the copy does not contain corrections for unique index violations, referential constraint violations, or check constraint violations because those corrections occur during the INDEXVAL, ENFORCE, and DISCARD phases.

To improve the performance of the recovery, take a full image copy of the page sets, and then quiesce them using the QUIESCE utility. This allows RECOVER TORBA to recover the page sets to the quiesce point with minimal use of the log.

A table space prefix for an image copy remains unchanged when you perform a point-in-time recovery using the FASTSWITCH YES option with CONCURRENT YES upon recovery. Here is an example of this procedure:

1. Create an image copy of the table space, for example, table space I0001, using CONCURRENT YES.
2. Reorganize the table space using FASTSWITCH YES. This changes the table space prefix to J0001.
3. Perform a point-in-time recovery with image copy I0001. After RECOVERY processing, the table space prefix is J0001.

***Authorization:*** Restrict use of TOCOPY and TORBA to personnel with a thorough knowledge of the DB2 recovery environment.

***Ensuring consistency:*** RECOVER TORBA and RECOVER TOCOPY can be used on a single:

- Partition of a partitioned table space
- Partition of a partitioning index space
- Page set of a simple table space

All page sets must be restored to the same level or the data will be inconsistent.

A table space and all of its indexes (or a table space set and all related indexes) should be recovered in the same RECOVER utility statement, specifying TORBA to identify a QUIESCE point or a common SHRLEVEL(REFERENCE) copy point. This action avoids placing indexes in the CHECK pending or RECOVER pending status. If the TORBA is not a common QUIESCE point or SHRLEVEL(REFERENCE) copy point for all objects, use the following procedure:

1. RECOVER table spaces to the log point.
2. Use concurrent REBUILD INDEX jobs to rebuild the indexes over each table space.

This procedure ensures that the table spaces and indexes are synchronized, and eliminates the need to run the CHECK INDEX utility.

Point-in-time recovery can cause table spaces to be placed in *check pending* status if they have table check constraints or referential constraints defined on them. When recovering tables involved in a referential constraint, you should recover all the table spaces involved in a constraint. This is the *table space set*. To avoid setting check pending, you must do both of the following:

- Recover the table space set to a quiesce point.

  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 “Violations of referential constraints” on page 443.

When recovering tables with LOB columns, you should recover the entire set of page sets, including the base table space, the LOB table spaces, and index spaces for the auxiliary indexes. Recovering a LOB table space to a prior point-in-time is similar to recovering a non-LOB table space to a prior point-in-time, with the following exceptions:

- The RECOVER utility set the auxiliary warning (AUXW) status for a LOB table space if it finds at least one invalid column during the LOGAPPLY phase.
- If you recover a LOB table space to a point-in-time that is not a QUIESCE point or to an image copy produced with SHRLEVEL CHANGE, the LOB table space is placed in check pending (CHKP) status.
- If you recover only the LOB table space to any previous point-in-time, the base table space is placed in auxiliary check pending (ACHKP) status, and the index space containing an index on the auxiliary table is placed in rebuild pending (RBDP) status.
- If you recover only the base table space to a point-in-time, the base table space is placed in auxiliary check pending (ACHKP) status.
- If you recover only the index space containing an index on the auxiliary table to a point-in-time, the index space is placed in check pending (CHKP) status.

See Part 2 of *DB2 Utility Guide and Reference* for detailed information about recovering a table space that contains LOB data.

***Compressed data:*** Use caution when recovering a single data set of a nonpartitioned page set to a prior point in time. If the data set being recovered was compressed with a different dictionary from the rest of the page set, then you can no longer read the data. For important information on loading and compressing data see the description of LOAD in Part 2 of *DB2 Utility Guide and Reference*.

## Recovery of dropped objects

The procedures described in this section can be used in the event that a table or table space is inadvertently dropped.

## Avoiding the problem

To avoid the problem of accidentally dropping tables, you can create a table with the clause WITH RESTRICT ON DROP. No one can drop the table, nor the table space or database containing the table, until the restriction on the table is removed. The ALTER TABLE statement includes a new clause to remove the restriction, as well as one to impose it.

## Procedures for recovery

The following terms are used throughout this discussion and are defined here:

| **Term** | **Meaning** |
|---|---|
| **DBID** | Database identifier |
| **OBID** | Data object identifier |
| **PSID** | Table space identifier |

To prepare for this procedure, it is a good idea to **run regular catalog reports** that include a list of all OBIDs in the subsystem. In addition, it is also very useful to have catalog reports listing dependencies on the table (such as referential constraints, indexes, and so on). After a table is dropped, this information disappears from the catalog.

If an OBID has been reused by DB2, you must run DSN1COPY to translate the OBIDs of the objects in the data set. However, this is unlikely; DB2 reuses OBIDs only when no image copies exist that contain data from that table.

## Recovery of an accidentally dropped table

Tables in a partitioned table space cannot be dropped without dropping the table space. If you have accidentally dropped a table space, see “Recovery of an accidentally dropped table space” on page 405.

To perform this procedure, you need a full image copy or a DSN1COPY file that contains the data from the dropped table.

For segmented table spaces, the image copy or DSN1COPY file must contain the table when it was active (that is, created). Because of the way space is reused for segmented table spaces, this procedure cannot be used if the table was not active when the image copy or DSN1COPY was made. For nonsegmented table spaces, the image copy or DSN1COPY file can contain the table when it was active or not active.

1. If you know the DBID, the PSID, the original OBID of the dropped table, and the OBIDs of all other tables contained in the table space, go to step 2.

   If you do not know all of the items listed above, use the following steps to find them. For later use with DSN1COPY, record the DBID, the PSID, and the OBIDs of all the tables contained in the table space, not just the dropped table.

   a. For the data set that contains the dropped table, run DSN1PRNT with the FORMAT option. Record the HPGOBID field in the header page and the PGSOBD field from the data records in the data pages.

      For the auxiliary table of a LOB table space, record the HPGROID field in the header page instead of PGSOBD field in the data pages.

      - Field HPGOBID is four bytes long and contains the DBID in the first two bytes and the PSID in the last two bytes.
      - Field HPGROID (for LOB table spaces) contains the OBID of the table. A LOB table space can contain only one table.
      - Field PGSOBD (for non-LOB table spaces) is two bytes long and contains the OBID of the table. If your table space contains more than one table, check for all OBIDs. In other words, search for all different PGSOBD fields. You need to specify all OBIDs from the data set as input for the DSN1COPY utility.

   b. Convert the hex values in the identifier fields to decimal so they can be used as input for the DSN1COPY utility.

2. Use the SQL CREATE statement to re-create the table and any indexes on the table.

3. To allow DSN1COPY to access the DB2 data set, stop the table space using the following command:

   ```
   -STOP DATABASE(database-name) SPACENAM(tablespace-name)
   ```

   This is necessary to ensure that all changes are written out and that no data updates occur during this procedure.

4. Find the new OBID for the table by querying the SYSIBM.SYSTABLES catalog table. The following statement returns the object ID (OBID) for the table:

   **Product-sensitive Programming Interface**

   ```
   SELECT NAME, OBID FROM SYSIBM.SYSTABLES
     WHERE NAME='table_name'
       AND CREATOR='creator_name';
   ```

   **End of Product-sensitive Programming Interface**

   This value is returned in decimal format, which is the format you need for DSN1COPY.

5. Run DSN1COPY with the OBIDXLAT and RESET options to perform the OBID translation and to copy the data from the full image copy data set, inline copy data set, or DSN1COPY file that contains the data from the dropped table into the original data set. Use the original OBIDs you recorded in step 1 and the new OBID you recorded in step 4 as the input records for the translation file (SYSXLAT). For more information about DSN1COPY, see Part 3 of *DB2 Utility Guide and Reference*.

   Be sure you have named the VSAM data sets correctly by checking messages DSN1998I and DSN1997I after DSN1COPY completes.

6. Start the table space for normal use using the following command:

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

7. Recover any indexes on the table.
8. Verify that you can access the table, including LOB columns, by executing SELECT statements to use the table.
9. Make a full image copy of the table space. See “Copying page sets and data sets” on page 391 for more information about the COPY utility.
10. Re-create the objects that are dependent on the table.

    As explained in “Implications of dropping a table” on page 66, when a table is dropped, all objects dependent on that table (synonyms, views, aliases, indexes, referential constraints, and so on) are dropped. Privileges granted for that table are dropped as well. Catalog reports or a copy of the catalog taken prior to the DROP TABLE can make this task easier.

## Recovery of an accidentally dropped table space

These procedures are for table spaces, including LOB table spaces, that were dropped accidentally. This can happen, for example, when all tables in an implicitly-created table space are dropped, or if someone unintentionally executes a DROP TABLESPACE statement for a particular table space.

When a table space is dropped, DB2 loses all information about the image copies of that table space. Although the image copy data set is not lost, locating it might require examination of image copy job listings or manually recorded information about the image copies.

Following are two separate procedures: one for user-managed data sets and one for DB2-managed data sets.

### User-managed data sets

In this procedure, you copy the data sets containing data from the dropped table space to redefined data sets using the “OBID translate” function of DSN1COPY.

1. Find the DBID for the database, the PSID for the dropped table space, and the OBIDs for the tables contained in the dropped table space. For information about how to do this, see step 1 of “Recovery of an accidentally dropped table” on page 403.
2. Rename the data set containing the dropped table space using the IDCAMS ALTER command. Do not forget to rename both the CLUSTER and DATA portion of the data set and to begin the data set name with the integrated catalog facility catalog name or alias.
3. Redefine the original DB2 VSAM data sets.

   Use the access method services LISTCAT command to obtain a list of data set attributes. The data set attributes on the redefined data sets must be the same as they were on the original data sets.
4. Use SQL CREATE statements to re-create the table space, tables and any indexes on the tables.
5. To allow DSN1COPY to access the DB2 data sets, stop the table space using the following command:

   ```
   -STOP DATABASE(database-name) SPACENAM(tablespace-name)
   ```

   This is necessary to prevent updates to the table space during this procedure in the event the table space has been left open.
6. Find the target OBIDs (the OBIDs for the tables and the PSID for the table space) by querying the SYSIBM.SYSTABLESPACE and SYSIBM.SYSTABLES

catalog tables.

**Product-sensitive Programming Interface**

The following statement returns the object ID for a table space; this is the PSID.

```
SELECT DBID, PSID FROM SYSIBM.SYSTABLESPACE
  WHERE NAME='tablespace_name' and DBNAME='database_name'
    AND CREATOR='creator_name';
```

The following statement returns the object ID for a table:

```
SELECT NAME, OBID FROM SYSIBM.SYSTABLES
  WHERE NAME='table_name'
    AND CREATOR='creator_name';
```

**End of Product-sensitive Programming Interface**

These values are returned in decimal format, which is the format you need for DSN1COPY.

7. Run DSN1COPY with the OBIDXLAT and RESET options to perform the OBID translation and to copy the data from the renamed VSAM data set containing the dropped table space to the redefined VSAM data set. Use of the RESET option prevents DB2 from marking data in the table space you restore as down level. Use the OBIDs you recorded from steps 1and 6 as the input records for the translation file (SYSXLAT). For more information about DSN1COPY, see Part 3 of *DB2 Utility Guide and Reference*.

   Be sure you have named the VSAM data sets correctly by checking messages DSN1998I and DSN1997I after DSN1COPY completes.
8. Start the table space for normal use by using the following command:

   ```
   -START DATABASE(database-name) SPACENAM(tablespace-name)
   ```
9. Recover all indexes on the table space.
10. Verify that you can access the table space, perhaps by executing SELECT statements to use each table.
11. Make a full image copy of the table space.

    See “Copying page sets and data sets” on page 391 for more information about the COPY utility.
12. Re-create the objects that are dependent on the table.

    See step 10 of “Recovery of an accidentally dropped table” on page 403 for more information.

## DB2-managed data sets

If a consistent full image copy or DSN1COPY file is available, DSN1COPY can be used to recover a dropped table space. To do this:

1. Find the original DBID for the database, the PSID for the table space, and the OBIDs of all tables contained in the dropped table space. For information on how to do this, see step 1 of “Recovery of an accidentally dropped table” on page 403.
2. Re-create the table space and all tables. This can be difficult for tables where either:
   - The table definition is not available.
   - The table is no longer required.

   In these cases, simply create a *dummy* table with any structure of columns.

3. Re-create auxiliary tables and indexes if a LOB table space has been dropped.
4. To allow DSN1COPY to access the DB2 data set, stop the table space with the following command:

   ```
   -STOP DATABASE(database-name) SPACENAM(tablespace-name)
   ```
5. Find the *new* DBID, PSID, and OBIDs by querying the DB2 catalog as described in step 6 of “User-managed data sets” on page 405.
6. Run DSN1COPY using OBIDXLAT and RESET options to perform the OBID translation and to copy the data from the full image copy data set, inline copy data set, or the DSN1COPY data set. Use the OBIDs you recorded from steps 1 and 5 as the input records for the translation file (SYSXLAT). For more information about DSN1COPY, see Part 3 of *DB2 Utility Guide and Reference*.

   Be sure you have named the VSAM data sets correctly by checking messages DSN1998I and DSN1997I after DSN1COPY completes.
7. Start the table space for normal use using the following command:

   ```
   -START DATABASE(database-name) SPACENAM(tablespace-name)
   ```
8. Drop all *dummy* tables. The row structure does not match the definition, so these tables cannot be used.
9. Reorganize the table space to remove all rows from dropped tables.
10. Recover all indexes on the table space.
11. Verify that you can access the table space, perhaps by executing SELECT statements to use each table.
12. Make a full image copy of the table space.

    See “Copying page sets and data sets” on page 391 for more information about the COPY utility.
13. Re-create the objects that are dependent on the table.

    See step 10 on page 405 of “Recovery of an accidentally dropped table” on page 403 for more information.

## Discarding SYSCOPY and SYSLGRNX records

Use the MODIFY utility to delete obsolete records from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX. To keep a table space and its indexes synchronized, the MODIFY utility deletes the SYSCOPY and SYSLGRNX records for the table space and its indexes defined with the COPY YES option.

1. Follow these steps of the procedure described under “Locating archive log data sets to delete” on page 343:
   a. “Resolve indoubt units of recovery” on page 344.
   b. “Find the startup log RBA” on page 344.
   c. “Find the minimum log RBA needed” on page 344. In that step, note the date of the earliest image copy you intend to keep.

   **What copies to keep:** The earliest image copies and log data sets you need for recovery to the present date are not necessarily the earliest ones you want to keep. If you foresee resetting the DB2 subsystem to its status at any earlier date, you also need the image copies and log data sets that allow you to recover to that date.

   If the most recent image copy of an object is damaged, the RECOVER utility seeks a backup copy. If there is no backup copy, or the backup is lost or damaged, RECOVER will use a previous image copy. It will continue searching until it finds an undamaged image copy or there are no more image copies. The process has important implications for keeping archive log data sets. At the very least, you need all log records since the most recent image copy; to protect against loss of data from damage to that copy, you need log records as far back as the earliest image copy you keep.

2. Run the MODIFY utility for each table space whose old image copies you want to discard, using the date of the earliest image copy you will keep. For example, you could enter:

   ```
   MODIFY RECOVERY TABLESPACE dbname.tsname
                   DELETE DATE date
   ```

   The DELETE DATE option removes records written earlier than the given date. You also can use DELETE AGE, to remove records older than a given number of days.

   You can delete SYSCOPY records for a single partition by naming it with the DSNUM keyword. That option does not delete SYSLGRNX records and does not delete SYSCOPY records that are later than the earliest point to which you can recover the entire table space. Thus, you can still recover by partition after that point.

   You cannot run the MODIFY utility on a table space that is in the *recovery pending* status.

# Chapter 22. Recovery scenarios

This chapter contains problem scenarios and the recommended procedures for restarting and recovering DB2. The following situations are described:

## IRLM failure

***Problem:*** The IRLM fails in a wait, loop, or abend.

***Symptom:*** The IRLM abends and the following message appears:

```
DXR122E irlmnm ABEND UNDER IRLM TCB/SRB IN MODULE xxxxxxxx
ABEND CODE zzzz
```

***System action:*** If the IRLM abends, DB2 terminates. If the IRLM waits or loops, then terminate the IRLM, and DB2 terminates automatically.

***System programmer action:*** None.

***Operator action:***

- Start the IRLM if you did not set it for automatic start when you installed DB2. (For instructions on starting the IRLM, see “Starting the IRLM” on page 281.)
- Start DB2. (For instructions, see “Starting DB2” on page 256.)
- Give the command /START SUBSYS *ssid* to connect IMS to DB2.

- Give the command DSNC STRT to connect CICS to DB2. (See “Connecting from CICS” on page 288.)

## MVS or power failure

***Problem:*** MVS or processor power fails.

***Symptom:*** No processing is occurring.

***System action:*** None.

***System programmer action:*** None.

***Operator action:***

1. IPL MVS and initialize the job entry subsystem.
2. If you normally run VTAM with DB2, start VTAM at this point.
3. Start the IRLM if you did not set it for automatic start when you installed DB2. (See “Starting the IRLM” on page 281.)
4. Start DB2. (See “Starting DB2” on page 256.)
5. Use the RECOVER POSTPONED command if postponed-abort units of recovery were reported after restarting DB2, and the AUTO option of the LIMIT BACKOUT field on installation panel DSNTIPN was not specified.
6. Restart IMS or CICS.
   a. IMS automatically connects and resynchronizes when it is restarted. (See “Connecting to the IMS control region” on page 295.)
   b. CICS automatically connects to DB2 if the CICS PLT contains an entry for the attach module DSNCCOM0. Alternatively, use the command DSNC STRT to connect the CICS attachment facility to DB2. (See “Connecting from CICS” on page 288.)

If you know that a power failure is imminent, it is a good idea to issue -STOP DB2 MODE(FORCE) to allow DB2 to come down cleanly before the power is interrupted. If DB2 is unable to stop completely before the power failure, the situation is no worse than if DB2 were still up.

## Disk failure

***Problem:*** A disk hardware failure occurs, resulting in the loss of an entire unit.

***Symptom:*** No I/O activity for the affected disk address. Databases and tables residing on the affected unit are unavailable.

***System action:*** None

***System programmer action:*** None

***Operator action:*** Attempt recovery by following these steps:

1. Assure that there are no incomplete I/O requests against the failing device. One way to do this is to force the volume off line by issuing the following MVS command:

   ```
   VARY xxx,OFFLINE,FORCE
   ```

   where *xxx* is the unit address.

To check disk status you can issue:

```
D U,DASD,ONLINE
```

A console message similar to the following is displayed after you have forced a volume offline:

```
  UNIT  TYPE  STATUS   VOLSER  VOLSTATE
  4B1   3390  O-BOX    XTRA02  PRIV/RSDNT
```

The disk unit is now available for service.

If you have previously set the I/O timing interval for the device class, the I/O timing facility should terminate all incomplete requests at the end of the specified time interval, and you can proceed to the next step without varying the volume off line. You can set the I/O timing interval either through the IECIOSxx MVS parameter library member or by issuing the MVS command

```
SETIOS MIH,DEV=devnum,IOTIMING=mm:ss.
```

For more information on the I/O timing facility, see *OS/390 MVS Initialization and Tuning Reference* and *OS/390 MVS System Commands*.

2. An authorized operator issues the following command to stop all databases and table spaces residing on the affected volume:

   ```
   -STOP DATABASE(database-name) SPACENAM(space-name)
   ```

   If the disk unit must be disconnected for repair, all databases and table spaces on all volumes in the disk unit must be stopped.

3. Select a spare disk pack and use ICKDSF to initialize from scratch a disk unit with a different unit address (*yyy*) and the same volser.

   ```
     // Job
     //ICKDSF   EXEC PGM=ICKDSF
     //SYSPRINT DD   SYSOUT=*
     //SYSIN    DD   *
          REVAL UNITADDRESS(yyy) VERIFY(volser)
   ```

   If you are initializing a 3380 or 3390 volume, use REVAL with the VERIFY parameter to ensure you are initializing the volume you want, or to revalidate the volume’s home address and record 0. Details are provided in *Device Support Facilities User's Guide and Reference*. Alternatively, use ISMF to initialize the disk unit.

4. Issue this MVS console command. *yyy* is the new unit address.

   ```
   VARY yyy,ONLINE
   ```

5. To check disk status you can issue:

   ```
   D U,DASD,ONLINE
   ```

   A console message similar to the following is displayed:

   ```
     UNIT  TYPE  STATUS   VOLSER  VOLSTATE
     7D4   3390  O        XTRA02  PRIV/RSDNT
   ```

6. Issue the following command to start all the appropriate databases and table spaces that had been stopped previously:

   ```
   -START DATABASE(database-name) SPACENAM(space-name)
   ```

7. Delete all table spaces (VSAM linear data sets) from the ICF catalog by issuing the following access method services command for each one of them:

   ```
   DELETE catnam.DSNDBC.dbname.tsname.y0001.A00x CLUSTER NOSCRATCH
   ```

where *y* can be either I or J.

Access method services commands are described in detail in *DFSMS/MVS: Access Method Services for VSAM Catalogs*.

8. For user-managed table spaces, the VSAM cluster and data components must be defined for the new volume by issuing the access method services DEFINE CLUSTER command with the data set name:

   ```
   catnam.DSNDBC.dbname.tsname.y0001.A00x
   ```

   where *y* can be either I or J, and *x* is C (for VSAM clusters) or D (for VSAM data components).

   This data set is the same as defined in Step 7. Detailed requirements for user-managed data sets are described in “Requirements for your own data sets” on page 34.

   For a user defined table space, the new data set must be defined before an attempt to recover it. Table spaces defined in storage groups can be recovered without prior definition.
9. Recover the table spaces using the RECOVER utility. Additional information and procedures for recovering data can be found in “Recovering page sets and data sets” on page 393.

## Application program error

***Problem:*** An application program placed a logically incorrect value in a table.

***Symptom:*** SQL SELECT returns unexpected data.

***System action:*** The system returns SQLCODE 0 for the SELECT statement, because the error was not in SQL or DB2, but in the application program. That error can be identified and corrected, but the data in the table is now inaccurate.

***System programmer action:*** You might be able to use RECOVER TORBA (or RECOVER TOLOGPOINT) to restore the database to a point before the error occurred, but there are many circumstances under which you must manually back out the changes introduced by the application. Among those are:

- Other applications changed the database after the error occurred. If you recover the table spaces modified by the bad application, you would lose all subsequent changes made by the other applications.
- There were DB2 checkpoints after the error occurred. In this case, you can use RECOVER TORBA to restore the data up to the last checkpoint before the error occurred, but all subsequent changes to the database are lost.

If you have a situation in which it makes sense to use RECOVER TORBA, you can use procedures similar to those that follow to back out the changes made by the bad application. For a discussion of RECOVER TORBA or TOLOGPOINT, see “Using RECOVER to restore data to a previous point in time” on page 400.

### Procedure 1: If you have established a quiesce point

1. Run the REPORT utility twice, once using the RECOVERY option and once using the TABLESPACESET option. On each run, specify the table space containing the inaccurate data. If you want to recover to the last quiesce point, specify the option CURRENT when running REPORT RECOVERY.

2. Examine the REPORT output to determine the RBA of the quiesce point.
3. Execute RECOVER TORBA (or TOLOGPOINT) with the RBA that you found, specifying the names of all related table spaces. Recovering all related table spaces to the same quiesce point prevents violations of referential constraints.

**Procedure 2: If you have not established a quiesce point**

If you use this procedure, you will lose any updates to the database that occurred after the last checkpoint before the application error occurred.

1. Run the DSN1LOGP stand-alone utility on the log scope available at DB2 restart, using the SUMMARY(ONLY) option. For instructions on running DSN1LOGP, see Part 3 of *DB2 Utility Guide and Reference*.
2. Determine the RBA of the most recent checkpoint before the first bad update occurred, from one of the following sources:
   - Message DSNR003I on the operator's console. It looks (in part) like this:

     ```
     DSNR003I RESTART  ..... PRIOR CHECKPOINT
                             RBA=000007425468
     ```

     The required RBA in this example is X'7425468'.

     This technique works only if there have been no checkpoints since the application introduced the bad updates.
   - Output from the print log map utility. You must know the time that the first bad update occurred. Find the last BEGIN CHECKPOINT RBA before that time.
3. Run DSN1LOGP again, using SUMMARY(ONLY) and specify the checkpoint RBA as the value of RBASTART. The output lists the work in the recovery log, including information about the most recent complete checkpoint, a summary of all processing occurring, and an identification of the databases affected by each active user. Sample output is shown in Figure 53 on page 484.
4. One of the messages in the output (identified as DSN1151I or DSN1162I) describes the unit of recovery in which the error was made. To find the unit of recovery, use your knowledge of the time the program was run (START DATE= and TIME=), the connection ID (CONNID=), authorization ID (AUTHID=), and plan name (PLAN=). In that message, find the starting RBA as the value of START=.
5. Execute RECOVER TORBA with the starting RBA you found in the previous step.
6. Recover any related table spaces or indexes to the same point in time.

***Operator action:*** None.

## IMS-related failures

This section includes scenarios for problems that can be encountered in the IMS environment:

"IMS control region (CTL) failure" on page 414
"Resolution of indoubt units of recovery" on page 414
"IMS application failure" on page 416

DB2 can be used in an XRF (Extended Recovery Facility) recovery environment with IMS. See "Extended recovery facility (XRF) toleration" on page 374 for more information on using XRF with IMS.

## IMS control region (CTL) failure

***Problem:*** The IMS control region fails.

***Symptom:***

- IMS waits, loops, or abends.
- DB2 attempts to send the following message to the IMS master terminal during an abend:

  ```
  DSNM002I  IMS/TM xxxx DISCONNECTED FROM SUBSYSTEM
           yyyy  RC=RC
  ```

  This message cannot be sent if the failure prevents messages from being displayed.
- DB2 does not send any messages related to this problem to the MVS console.

***System action:***

- DB2 detects that IMS has failed.
- DB2 either backs out or commits work in process.
- DB2 saves indoubt units of recovery. (These must be resolved at reconnection time.)

***System programmer action:*** None.

***Operator action:***

1. Use normal IMS restart procedures, which include starting IMS by issuing the MVS START IMS command.
2. The following results occur:
   - All DL/I and DB2 updates that have not been committed are backed out.
   - IMS is automatically reconnected to DB2.
   - IMS passes the recovery information for each entry to DB2 through the IMS attachment facility. (IMS indicates whether to commit or roll back.)
   - DB2 resolves the entries according to IMS instructions.

## Resolution of indoubt units of recovery

This section describes two different problems.

### Problem 1

There are unresolved indoubt units of recovery. When IMS connects to DB2, DB2 has one or more indoubt units of recovery that have not been resolved.

***Symptom:*** If DB2 has indoubt units of recovery that IMS did not resolve, the following message is issued at the IMS master terminal:

```
DSNM004I  RESOLVE INDOUBT ENTRY(S) ARE OUTSTANDING FOR
          SUBSYSTEM xxxx
```

When this message is issued, IMS was either cold started or it was started with an incomplete log tape. This message could also be issued if DB2 or IMS had an abend due to a software error or other subsystem failure.

***System action:***

- The connection remains active.
- IMS applications can still access DB2 databases.
- Some DB2 resources remain locked out.

If the indoubt thread is not resolved, the IMS message queues can start to back up. If the IMS queues fill to capacity, IMS terminates. Therefore, users must be aware of this potential difficulty and must monitor IMS until the indoubt units of work are fully resolved.

***System programmer action:***

1. Force the IMS log closed using /DBR FEOV, and then archive the IMS log. Use the command DFSERA10 to print the records from the previous IMS log tape for the last transaction processed in each dependent region. Record the PSB and the commit status from the X'37' log containing the recovery ID.
2. Run the DL/I batch job to back out each PSB involved that has not reached a commit point. The process might take some time because transactions are still being processed. It might also lock up a number of records, which could impact the rest of the processing and the rest of the message queues.
3. Enter the DB2 command DISPLAY THREAD (*imsid*) TYPE (INDOUBT).
4. Compare the NIDs (IMSID + OASN in hexadecimal) displayed in the -DISPLAY THREAD messages with the OASNs (4 bytes decimal) shown in the DFSERA10 output. Decide whether to commit or roll back.
5. Use DFSERA10 to print the X'5501FE' records from the current IMS log tape. Every unit of recovery that undergoes indoubt resolution processing is recorded; each record with an 'IDBT' code is still indoubt. Note the correlation ID and the recovery ID, because they will be used during step 6.
6. Enter the following DB2 command, choosing to commit or roll back, and specifying the correlation ID:

   ```
   -RECOVER INDOUBT (imsid) ACTION(COMMIT|ABORT) NID (nid)
   ```

   If the command is rejected because there are more network IDs associated, use the same command again, substituting the recovery ID for the network ID.

(For a description of the OASN and the NID, see “Duplicate correlation IDs” on page 299.)

***Operator action:*** Contact the system programmer.

## Problem 2

Committed units of recovery should be aborted. At the time IMS connects to DB2, DB2 has committed one or more indoubt units of recovery that IMS says should be rolled back.

***Symptom:*** By DB2 restart time, DB2 has committed and rolled back those units of recovery about which DB2 was not indoubt. DB2 records those decisions, and at connect time, verifies that they are consistent with the IMS/VS decisions.

An inconsistency can occur when the DB2 -RECOVER INDOUBT command is used before IMS attempted to reconnect. If this happens, the following message is issued at the IMS master terminal:

```
DSNM005I  IMS/TM RESOLVE INDOUBT PROTOCOL PROBLEM WITH
          SUBSYSTEM xxxx
```

Because DB2 tells IMS to retain the inconsistent entries, the following message is issued when the resolution attempt ends:

```
DFS3602I  xxxx SUBSYSTEM RESOLVE-INDOUBT FAILURE,
          RC=yyyy
```

***System action:***

- The connection between DB2 and IMS remains active.
- DB2 and IMS continue processing.
- No DB2 locks are held.
- No units of work are in an incomplete state.

***System programmer action:*** Do not use the DB2 command RECOVER INDOUBT. The problem is that DB2 was *not* indoubt but should have been.

Database updates have most likely been committed on one side (IMS or DB2) and rolled back on the other side. (For a description of the OASN and the NID, see “Duplicate correlation IDs” on page 299.)

1. Enter the IMS command /DISPLAY OASN SUBSYS DB2 to display the IMS list of units of recovery that need to be resolved. The /DISPLAY OASN SUBSYS DB2 command produces the OASNs in a decimal format, not a hexadecimal format.
2. Issue the IMS command /CHANGE SUBSYS DB2 RESET to reset all the entries in the list. (No entries are passed to DB2.)
3. Use DFSERA10 to print the log records recorded at the time of failure and during restart. Look at the X'37', X'56', and X'5501FE' records at reconnect time. Notify the IBM support center about the problem.
4. Determine what the inconsistent unit of recovery was doing by using the log information, and manually make the DL/I and DB2 databases consistent.

***Operator action:*** None.

## IMS application failure

This section describes two different problems.

### Problem 1

An IMS application abends.

***Symptom:*** The following messages appear at the IMS master terminal and at the LTERM that entered the transaction involved:

```
DFS555  - TRAN tttttttt ABEND (SYSIDssss);
          MSG IN PROCESS: xxxx (up to 78 bytes of data) timestamp
DFS555A - SUBSYSTEM xxxx OASN yyyyyyyyyyyyyyyy STATUS COMMIT|ABORT
```

***System action:***

The failing unit of recovery is backed out by both DL/I and DB2.
The connection between IMS and DB2 remains active.

***System programmer action:*** None.

***Operator action:*** If you think the problem was caused by a user error, refer to Part 2 of *DB2 Application Programming and SQL Guide*. For procedures to diagnose DB2 problems, rather than user errors, refer to Part 3 of *DB2 Diagnosis Guide and Reference*. If necessary, contact the IBM support center for assistance.

### Problem 2

DB2 has failed or is not running.

***Symptom:*** One of the following status situations exists:

- If you specified error option Q, the program terminates with a U3051 user abend completion code.

- If you specified error option A, the program terminates with a U3047 user abend completion code.

In both cases, the master terminal receives a message (IMS message number DFS554), and the terminal involved also receives a message (DFS555).

***System action:*** None.

***System programmer action:*** None.

***Operator action:***

1. Restart DB2.
2. Follow the standard IMS procedures for handling application abends.

## CICS-related failures

This section includes scenarios for problems that can be encountered in the CICS environment:

“CICS application failure”
“CICS is not operational”
“CICS cannot connect to DB2” on page 418
“Manually recovering CICS indoubt units of recovery” on page 419
“CICS attachment facility failure” on page 422

DB2 can be used in an XRF (Extended Recovery Facility) recovery environment with CICS. See “Extended recovery facility (XRF) toleration” on page 374 for more information on using XRF with CICS.

## CICS application failure

***Problem:*** A CICS application abends.

***Symptom:*** The following message is sent to the user’s terminal.

```
DFH2206 TRANSACTION tranid ABEND abcode BACKOUT SUCCESSFUL
```

*tranid* can represent any abending CICS transaction and *abcode* is the abend code.

***System action:***

The failing unit of recovery is backed out in both CICS and DB2.
The connection remains.

***System programmer action:*** None.

***Operator action:***

1. For information about the CICS attachment facility abend, refer to Part 2 of *DB2 Messages and Codes*.
2. For an *AEY9* abend, start the CICS attachment facility.
3. For an *ASP7* abend, determine why the CICS SYNCPOINT was unsuccessful.
4. For other abends, see *DB2 Diagnosis Guide and Reference* or *CICS/ESA Problem Determination Guide* for diagnostic procedures.

## CICS is not operational

***Problem:*** CICS is not operational.

***Symptom:*** More than one symptom is possible.

- CICS waits or loops.

  Because DB2 cannot detect a wait or loop in CICS, you must find the origin of the wait or the loop. The origin can be in CICS, CICS applications, or in the CICS attachment facility. For diagnostic procedures for waits and loops, see Part 2 of *DB2 Diagnosis Guide and Reference*.
- CICS abends.
  - CICS issues messages indicating an abend occurred and requests abend dumps of the CICS region. See *CICS/ESA Problem Determination Guide* for more information.
  - If threads are connected to DB2 when CICS terminates, DB2 issues message DSN3201I. The message indicates that DB2 end-of-task (EOT) routines have been run to clean up and disconnect any connected threads.

***System action:*** DB2 does the following:
Detects the CICS failure.
Backs out inflight work.
Saves indoubt units of recovery to be resolved when CICS is reconnected.

***Operator action:***

1. Correct the problem that caused CICS to terminate abnormally.
2. Do an emergency restart of CICS. The emergency restart accomplishes the following:
   - Backs out inflight transactions that changed CICS resources
   - Remembers the transactions with access to DB2 that might be indoubt.
3. Start the CICS attachment facility by entering the appropriate command for your release of CICS. See “Connecting from CICS” on page 288. The CICS attachment facility does the following:
   - Initializes and reconnects to DB2.
   - Requests information from DB2 about the indoubt units of recovery and passes the information to CICS.
   - Allows CICS to resolve the indoubt units of recovery.

## CICS cannot connect to DB2

***Problem:*** The CICS attachment facility cannot connect to DB2.

***Symptom:***

- CICS remains operative, but the CICS attachment facility abends.
- The CICS attachment facility issues a message giving the reason for the connection failure, or it requests an X'04E' dump.
- The reason code in the X'04E' dump gives the reason for failure.
- CICS issues message DFH2206 indicating that the CICS attach facility has terminated abnormally with the DSNC abend code.
- CICS application programs trying to access DB2 while the CICS attachment facility is inactive are abnormally terminated. The code AEY9 is issued.

***System Action:*** CICS backs out the abnormally terminated transaction and treats it like an application abend.

***Operator action:***

1. Start the CICS attachment facility by entering the appropriate command for your release of CICS. See “Connecting from CICS” on page 288.

2. The CICS attachment facility initializes and reconnects to DB2.
3. The CICS attachment facility requests information about the indoubt units of recovery and passes the information to CICS.
4. CICS resolves the indoubt units of recovery.

## Manually recovering CICS indoubt units of recovery

When the attachment facility abends, CICS and DB2 build indoubt lists either dynamically or during restart, depending on the failing subsystem.

For CICS, a DB2 unit of recovery could be indoubt if the forget entry (X'FD59') of the task-related installation exit is absent from the CICS system journal. The indoubt condition applies only to the DB2 UR, because CICS will have already committed or backed out any changes to its resources.

A DB2 unit of recovery is indoubt for DB2 if an End Phase 1 is present and the Begin Phase 2 is absent.

***Problem:*** When CICS connects to DB2, there are one or more indoubt units of recovery that have not been resolved.

***Symptom:*** One of the following messages is sent to the user-named CICS destination specified in the ERRDEST field in the resource control table (RCT): DSN2001I, DSN2034I, DSN2035I, or DSN2036I.

***System action:*** The system action is summarized in Table 68:

*Table 68. CICS abnormal indoubt unit of recovery situations*

| Message ID | Meaning |
|---|---|
| DSN2001I | The named unit of recovery cannot be resolved by CICS because CICS was cold started. The CICS attachment facility continues the startup process. |
| DSN2034I | The named unit of recovery is not indoubt for DB2, but is indoubt according to CICS log information. The reason is probably a CICS restart with the wrong tape. It could also be caused by a DB2 restart to a prior point in time. |
| DSN2035I | The named unit of recovery is indoubt for DB2, but is not in the CICS indoubt list. This is most likely due to an incorrect CICS restart. The CICS attachment facility continues the startup process and provides a transaction dump. It could also be caused by a DB2 restart to a prior point in time. |
| DSN2036I | CICS indicates roll back for the named unit of recovery, but DB2 has already committed the unit of recovery. The CICS attachment facility continues the startup process. |

CICS retains details of indoubt units of recovery that were not resolved during connection start up. An entry is purged when it no longer appears on the list presented by DB2 or, when present, DB2 solves it.

***System programmer action:*** Any indoubt unit of recovery that CICS cannot resolve must be resolved manually by using DB2 commands. This manual procedure should be used rarely within an installation, because it is required only where operational errors or software problems have prevented automatic resolution. *Any inconsistencies found during indoubt resolution must be investigated.*

To recover an indoubt unit, follow these steps:

**Step 1: Obtain a list of the indoubt units of recovery from DB2:**

Issue the following command:

```
-DISPLAY THREAD (connection-name) TYPE (INDOUBT)
```

You will receive the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV406I - INDOUBT THREADS -
COORDINATOR      STATUS      RESET  URID         AUTHID
coordinator-name status      yes/no urid        authid
DISPLAY INDOUBT REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

The *corr_id* (correlation ID) for CICS TS 1.1 and previous releases of CICS consists of:

**Byte 1**
Connection type: G = group, P = pool

**Byte 2**
Thread type: T = transaction (TYPE=ENTRY), G = group, C = command (TYPE=COMD)

**Bytes 3, 4**
Thread number

**Bytes 5-8**
Transaction ID

The *corr_id* (correlation ID) for CICS TS 1.2 and subsequent releases of CICS consists of:

**Bytes 1-4**
Thread type: COMD, POOL, or ENTR

**Bytes 5-8**
Transaction ID

**Bytes 9-12**
Unique thread number

It is possible for two threads to have the same correlation ID when the connection has been broken several times and the indoubt units of recovery have not been resolved. In this case, the network ID (NID) must be used instead of the correlation ID to uniquely identify indoubt units of recovery.

The network ID consists of the CICS connection name and a unique number provided by CICS at the time the syncpoint log entries are written. This unique number is an eight-byte store clock value that is stored in records written to both the CICS system log and to the DB2 log at syncpoint processing time. This value is referred to in CICS as the *recovery token*.

**Step 2: Scan the CICS log for entries related to a particular unit of recovery:** To do this, search the CICS log, looking for a PREPARE record (JCRSTRIDX'F959'), for the task-related installation where the recovery token field (JCSRMTKN) equals the value obtained from the network-ID. The network ID is supplied by DB2 in the DISPLAY THREAD command output.

Locating the prepare log record in the CICS log for the indoubt unit of recovery provides the CICS task number. All other entries on the log for this CICS task can be located using this number.

CICS journal print utility DFHJUP can be used when scanning the log. See *CICS for MVS/ESA Operations and Utilities Guide* for details on how to use this program.

**Step 3: Scan the DB2 log for entries related to a particular unit of recovery:** To do this, scan the DB2 log to locate the End Phase 1 record with the network ID required. Then use the URID from this record to obtain the rest of the log records for this unit of recovery.

When scanning the DB2 log, note that the DB2 start up message DSNJ099I provides the start log RBA for this session.

The DSN1LOGP utility can be used for that purpose. See Part 3 of *DB2 Utility Guide and Reference* for details on how to use this program.

**Step 4: If needed, do indoubt resolution in DB2:** DB2 can be directed to take the recovery action for an indoubt unit of recovery using a DB2 RECOVER INDOUBT command. Where the correlation ID is unique, use the following command:

```
DSNC -RECOVER INDOUBT (connection-name)
              ACTION (COMMIT/ABORT)
              ID (correlation-id)
```

If the transaction is a pool thread, use the value of the correlation ID (*corr_id*) returned by DISPLAY THREAD for *thread#.tranid* in the command RECOVER INDOUBT. In this case, the first letter of the correlation ID is P. The transaction ID is in characters five through eight of the correlation ID.

If the transaction is assigned to a group (group is a result of using an entry thread), use *thread#.groupname* instead of *thread#.tranid*. In this case, the first letter of the correlation ID is a G and the group name is in characters five through eight of the correlation ID. *groupname* is the first transaction listed in a group.

Where the correlation ID is not unique, use the following command:

```
DSNC -RECOVER INDOUBT (connection-name)
              ACTION (COMMIT/ABORT)
              NID (network-id)
```

When two threads have the same correlation ID, use the NID keyword instead of the ID keyword. The NID value uniquely identifies the work unit.

To recover all threads associated with *connection-name*, omit the ID option.

The command results in either of the following messages to indicate whether the thread is committed or rolled back:

```
DSNV414I - THREAD thread#.tranid COMMIT SCHEDULED
DSNV415I - THREAD thread#.tranid ABORT SCHEDULED
```

When performing indoubt resolution, note that CICS and the attachment facility are not aware of the commands to DB2 to commit or abort indoubt units of recovery, because only DB2 resources are affected. However, CICS keeps details about the indoubt threads that could not be resolved by DB2. This information is purged either when the list presented is empty, or when the list does not include a unit of recovery that CICS remembers.

***Operator action:*** Contact the system programmer.

## CICS attachment facility failure

***Problem:*** The CICS attachment facility abends, or a CICS attachment thread subtask abends. CICS and DB2 remain active.

***Symptom:***

- If the main subtask abends, an abend dump is requested. The contents of the dump indicate the cause of the abend. When the dump is issued, shutdown of the CICS attachment facility begins.
- If a thread subtask terminates abnormally, an X'04E' dump is issued and the CICS application abends with a DSNC dump code. The X'04E' dump should show the cause of the abend. The CICS attachment facility remains active.

***System action:***

- The CICS attachment facility shuts down if there is a main subtask abend.
- The matching CICS application abends with a DSNC dump code if a thread subtask abends.

***System programmer action:*** None.

***Operator action:*** Correct the problem that caused the abend by analyzing the CICS formatted transaction dump or subtask SNAP dump. For more information about analyzing these dumps, see Part 2 of *DB2 Messages and Codes*. If the CICS attachment facility shuts down, use CICS commands to stop the execution of any CICS-DB2 applications.

## Subsystem termination

***Problem:*** Subsystem termination has been started by DB2 or by an operator cancel.

***Symptom:*** Subsystem termination occurs. Usually some specific failure is identified by DB2 messages, and the following messages appear.

On the MVS console:

```
DSNV086E - DB2 ABNORMAL TERMINATION REASON=XXXXXXXX
DSN3104I - DSN3EC00 - TERMINATION COMPLETE
DSN3100I - DSN3EC00 - SUBSYSTEM ssnm READY FOR -START COMMAND
```

On the IMS master terminal:

```
DSNM002I   IMS/TM xxxx DISCONNECTED FROM SUBSYSTEM
           yyyy  RC=rc
```

On the CICS transient data error destination defined in the RCT:

```
DSNC2025I - THE ATTACHMENT FACILITY IS INACTIVE
```

***System action:***

- IMS and CICS continue.
- In-process CICS and IMS applications receive SQLCODE -923 (SQLSTATE '57015') when accessing DB2.

  In most cases, if an IMS or CICS application program is running when a -923 SQLCODE is returned, an abend occurs. This is because the application program generally terminates when it receives a -923 SQLCODE. To terminate, some synchronization processing occurs (such as a commit). If DB2 is not operational when synchronization processing is attempted by an application program, the application program abends. In-process applications can abend with an abend code X'04F'.

- New IMS applications are handled according to the error options.
  - For option R, SQL return code -923 is sent to the application, and IMS pseudo abends.
  - For option Q, the message is enqueued again and the transaction abends.
  - For option A, the message is discarded and the transaction abends.
- New CICS applications are handled as follows:
  - If the CICS attachment facility has not terminated, the application receives a -923 SQLCODE.
  - If the CICS attachment facility has terminated, the application abends (code AEY9).

***Operator action:***

1. Restart DB2 by issuing the command START DB2.
2. Reestablish the IMS connection by issuing the IMS command /START SUBSYS DB2.
3. Reestablish the CICS connection by issuing the CICS attachment facility command DSNC STRT.

***System programmer action:***

1. Use the IFCEREP1 service aid to obtain a listing of the SYS1.LOGREC data set containing the SYS1.LOGREC entries. (For more information about this service aid, refer to the MVS diagnostic techniques publication about SYS1.LOGREC.)
2. If the subsystem termination was due to a failure, collect material to determine the reason for failure (console log, dump, and SYS1.LOGREC).

## DB2 system resource failures

This section includes scenarios for problems that can be encountered in the DB2 environment:

“Active log failure”
“Archive log failure” on page 427
“BSDS failure” on page 429
“Recovering the BSDS from a backup copy” on page 431

## Active log failure

This section covers some of the more likely active log problems. Problems not covered here include the following:

- Active log dynamic allocation problems are indicated by message DSNJ103I at startup time.
- Active log open/close problems are indicated by message DSNJ104I.

Those problems are covered in “Chapter 23. Recovery from BSDS or log failure during restart” on page 475.

### Problem 1 - Out of space in active logs

The available space in the active log is finite and can be exhausted. It can fill to capacity for one of several reasons, such as delays in offloading and excessive logging.

***Symptom:*** An out of space condition on the active log has very serious consequences. When the active log becomes full, the DB2 subsystem cannot do any work that requires writing to the log until an offload is completed.

Due to the serious implications of this event, the DB2 subsystem issues the following warning message when the last available active log data set is 5 percent full and reissues the message after each additional 5 percent of the data set space is filled. Each time the message is issued, the offload process is started. IFCID trace record 0330 is also issued if statistics class 3 is active.

```
DSNJ110E - LAST COPYn ACTIVE LOG DATA SET IS nnn PERCENT FULL
```

If the active log fills to capacity, after having switched to single logging, the following message is issued, and an offload is started. The DB2 subsystem then halts processing until an offload has completed.

```
DSNJ111E - OUT OF SPACE IN ACTIVE LOG DATA SETS
```

Corrective action is required before DB2 can continue processing.

***System action:*** DB2 waits for an available active log data set before resuming normal DB2 processing. Normal shutdown, with either QUIESCE or FORCE, is not possible because the shutdown sequence requires log space to record system events related to shutdown (for example, checkpoint records).

***Operator action:*** Make sure offload is not waiting for a tape drive. If it is, mount a tape and DB2 will process the offload command.

If you are uncertain about what is causing the problem, enter the following command:

```
-ARCHIVE LOG CANCEL OFFLOAD
```

This command causes DB2 to restart the offload task. This might solve the problem.

If this command does not solve the problem, you must determine the cause of the problem and then reissue the command again. If the problem cannot be solved quickly, have the system programmer define additional active logs.

***System programmer action:*** Additional active log data sets can permit DB2 to continue its normal operation while the problem causing the offload failures is corrected.

1. Use the MVS command CANCEL command to bring DB2 down.
2. Use the access method services DEFINE command to define new active log data sets. Run utility DSNJLOGF to initialize the new active log data sets.

   To minimize the number of offloads taken per day in your installation, consider increasing the size of the active log data sets.
3. Define the new active log data sets in the BSDS by using the change log inventory utility (DSNJU003). For additional details, see Part 3 of *DB2 Utility Guide and Reference* .
4. Restart DB2. Off-load is started automatically during startup, and restart processing occurs.

## Problem 2 - Write I/O error on active log data set

***Symptom:*** The following message appears:

```
DSNJ105I - csect-name LOG WRITE ERROR DSNAME=..., LOGRBA=...,
           ERROR STATUS=ccccffss
```

***System action:***

Marks the failing log data set TRUNCATED in the BSDS.
Goes on to the next available data set.
If dual active logging is used, truncates the other copy at the same point.
The data in the truncated data set is offloaded later, as usual.
The data set is not *stopped*. It is reused on the next cycle. However, if there is a DSNJ104 message indicating that there is a CATUPDT failure, then the data set is marked “stopped”.

***System programmer action:*** If you get the DSNJ104 message indicating CATUPDT failure, you must use access method services and the change log inventory utility (DSNJU003) to add a replacement data set. This requires that you bring DB2 down. When you do this depends on how widespread the problem is.

- If the problem is localized and does not affect your ability to recover from any further problems, you can wait until the earliest convenient time.
- If the problem is widespread (perhaps affecting an entire set of active log data sets), take DB2 down after the next offload.

For instructions on using the change log inventory utility, see Part 3 of *DB2 Utility Guide and Reference*.

## Problem 3 - Dual logging is lost

***Symptom:*** The following message appears:

```
DSNJ004I - ACTIVE LOG COPYn INACTIVE, LOG IN SINGLE MODE,
           ENDRBA=...
```

Having completed one active log data set, DB2 found that the subsequent (COPY n) data sets were not offloaded or were marked “stopped”.

***System action:*** Continues in single mode until offloading completes, then returns to dual mode. If the data set is marked “stopped”, however, then intervention is required.

***System programmer action:*** Check that offload is proceeding and is not waiting for a tape mount. It might be necessary to run the print log map utility to determine the status of all data sets.

If there are “stopped” data sets, you must use IDCAMS to delete the data sets, and then re-add them using the change log inventory utility (DSNJU003). See Part 3 of *DB2 Utility Guide and Reference* for information about using the change log inventory utility.

## Problem 4 - I/O errors while reading the active log

***Symptom:*** The following message appears:

```
DSNJ106I - LOG READ ERROR DSNAME=..., LOGRBA=...,
           ERROR STATUS=ccccffss
```

***System action:***

- If the error occurs during offload, offload tries to pick the RBA range from a second copy.
  - If no second copy exists, the data set is stopped.
  - If the second copy also has an error, only the original data set that triggered the offload is stopped. Then the archive log data set is terminated, leaving a discontinuity in the archived log RBA range.
  - The following message is issued.

```
DSNJ124I - OFFLOAD OF ACTIVE LOG SUSPENDED FROM RBA xxxxxx
           TO RBA xxxxxx DUE TO I/O ERROR
```

  – If the second copy is satisfactory, the first copy is not stopped.

- If the error occurs during recovery, DB2 provides data from specific log RBAs requested from another copy or archive. If this is unsuccessful, recovery fails and the transaction cannot complete, but no log data sets are stopped. However, the table space being recovered is not accessible.

***System programmer action:*** If the problem occurred during offload, determine which databases are affected by the active log problem and take image copies of those. Then proceed with a new log data set.

Also, you can use IDCAMS REPRO to archive as much of the stopped active log data set as possible. Then run the change log inventory utility to notify the BSDS of the new archive log and its log RBA range. Repairing the active log does not solve the problem, because offload does not go back to unload it.

If the active log data set has been stopped, it is not used for logging. The data set is not deallocated; it is still used for reading.

If the data set is not stopped, an active log data set should nevertheless be replaced if persistent errors occur. The operator is not told explicitly whether the data set has been stopped. To determine the status of the active log data set, run the print log map utility (DSNJU004). For more information on the print log map utility, see Part 3 of *DB2 Utility Guide and Reference*.

To replace the data set, take the following steps:

1. Be sure the data is saved.

   If you have dual active logs, the data is saved on the other active log and it becomes your *new data set*. Skip to step 4.

   If you have not been using dual active logs, take the following steps to determine whether the data set with the error has been offloaded:

   a. Use the print log map to list information about the archive log data sets from the BSDS.

   b. Search the list for a data set whose RBA range includes the range of the data set with the error.

2. If the data set with the error has been offloaded (that is, if the value for High RBA Off-loaded in the print log map output is greater than the RBA range of the data set with the error), you need to manually add a new archive log to the BSDS using the change log inventory utility (DSNJU003). Use IDCAMS to define a new log having the same LRECL and BLKSIZE values as that defined in DSNZP*xxx*. You can use the access method services REPRO command to copy a data set with the error to the new archive log. If the archive log is not cataloged, DB2 can locate it from the UNIT and VOLSER values in the BSDS.

3. If an active log data set has been stopped, an RBA range has not been offloaded; copy from the data set with the error to a new data set. If further I/O errors prevent you from copying the entire data set, a gap occurs in the log and restart might fail, though the data still exists and is not overlaid. If this occurs, see “Chapter 23. Recovery from BSDS or log failure during restart” on page 475.

4. Stop DB2, and use change log inventory to update information in the BSDS about the data set with the error.

   a. Use DELETE to remove information about the bad data set.

b. Use NEWLOG to name the new data set as the new active log data set and to give it the RBA range that was successfully copied.

   The DELETE and NEWLOG operations can be performed by the same job step; put the DELETE statement before the NEWLOG statement in the SYSIN input data set. This step will clear the stopped status and DB2 will eventually archive it.

5. Delete the data set in error by using access method services.
6. Redefine the data set so you can write to it. Use access method services DEFINE command to define the active log data sets. Run utility DSNJLOGF to initialize the active log data sets. If using dual logs, use access method services REPRO to copy the good log into the redefined data set so that you have two consistent, correct logs again.

## Archive log failure

This section covers some of the more likely archive log problems. Problems not covered here include archive log open/close problems that are indicated by the message DSNJ104I. Most archive log problems are described in “Chapter 23. Recovery from BSDS or log failure during restart” on page 475.

### Problem 1 - Allocation problems

***Symptom:*** The following message appears:

```
DSNJ103I - csect-name LOG ALLOCATION ERROR DSNAME=dsname,
ERROR STATUS=eeeeiiii, SMS REASON CODE=ssssssss
```

MVS dynamic allocation provides the ERROR STATUS. If the allocation was for offload processing, the following is also displayed.

```
DSNJ115I - OFFLOAD FAILED, COULD NOT ALLOCATE AN ARCHIVE DATA SET
```

***System action:*** One of the following occurs:

- The RECOVER utility is executing and requires an archive log. If neither log can be found or used, recovery fails.
- The active log became full and an offload was scheduled. Off-load tries again the next time it is triggered. The active log does not wrap around; therefore, if there are no more active logs, data is not going to be lost.
- The input is needed for restart, which fails; refer to “Chapter 23. Recovery from BSDS or log failure during restart” on page 475.

***Operator action:*** Check the allocation error code for the cause of the problem and correct it. Ensure that drives are available and run the recovery job again. Caution must be exercised if a DFP/DFSMS ACS user-exit filter has been written for an archive log data set, because this can cause the DB2 subsystem to fail on a device allocation error attempting to read the archive log data set.

### Problem 2 - Write I/O errors during archive log offload

***Symptom:*** No specific DB2 message is issued for write I/O errors. Only an MVS error recovery program message appears. If you get DB2 message DSNJ128I, the offload task has abended and you should consult Part 2 of *DB2 Messages and Codes*.

***System action:***

- Off-load abandons that output data set (no entry in BSDS).
- Off-load dynamically allocates a new archive and restarts offloading from the point at which it was previously triggered. If there is dual archiving, the second copy waits.

- If an error occurs on the new data set, the following occurs.
  - If in dual archive mode, message DSNJ114I is generated and the offload processing changes to single mode.

    ```
    DSNJ114I - ERROR ON ARCHIVE DATA SET, OFFLOAD CONTINUING
              WITH ONLY ONE ARCHIVE DATA SET BEING GENERATED
    ```
  - If in single mode, it abandons the output data set. Another attempt to offload this RBA range is made the next time offload is triggered.
  - The active log does not wrap around; if there are no more active logs, data is not lost.

***Operator action:*** Ensure that offload is allocated on a good drive and control unit.

## Problem 3 - Read I/O errors on archive data set during recover

***Symptom:*** No specific DB2 message is issued, only the MVS error recovery program message appears.

***System action:***

- If a second copy exists, it is allocated and used.
- If a second copy does not exist, recovery fails.

***Operator action:*** If you are recovering from tape, try recovering using a different drive. If this does not work, contact the system programmer.

***System programmer action:*** The only option is to recover to the last image copy or the last quiesce point RBA. See Part 2 of *DB2 Utility Guide and Reference* for more information about using the RECOVER utility.

## Problem 4 - Insufficient disk space for offload processing

***Symptom:*** While offloading the active log data sets to disk, DB2 offload processing terminates unexpectedly. DB2 does not issue any specific message other than:

```
DSNJ128I - LOG OFFLOAD TASK FAILED FOR ACTIVE LOG nnnnn
```

The failure is preceded by MVS ABEND messages IEC030I, IEC031I, or IEC032I.

***System action:*** DB2 deallocates the data set on which the error occurred. If in dual archive mode, DB2 changes to single archive mode and continues the offload. If the offload cannot compete in single archive mode, the active log data sets cannot be offloaded, and the status of the active log data sets remains NOTREUSEABLE. Another attempt to offload the RBA range of the active log data sets is made the next time offload is invoked.

***System programmer action:*** If DB2 is operating with restricted active log resources (see message DSNJ110E), quiesce the DB2 subsystem to restrict logging activity until the MVS ABEND is resolved.

This message is generated for a variety of reasons. When accompanied by the MVS abends mentioned above, the most likely failures are as follows:

- The size of the archive log data set is too small to contain the data from the active log data sets during offload processing. All secondary space allocations have been used. This condition is normally accompanied by MVS ABEND message IEC030I.

  To solve the problem, increase the primary or secondary allocations (or both) for the archive log data set in DSNZP*xxx*. Another option is to reduce the size of the active log data set. If the data to be offloaded is particularly large, you can mount another online storage volume or make one available to DB2. Modifications to DSNZP*xxx* require that you stop and start DB2 to take effect.

- All available space on the disk volumes to which the archive data set is being written has been exhausted. This condition is normally accompanied by MVS ABEND message IEC032I.

  To solve the problem, make space available on the disk volumes, or make available another online storage volume for DB2. Then issue the DB2 command ARCHIVE LOG CANCEL OFFLOAD to get DB2 to retry the offload.

- The primary space allocation for the archive log data set (as specified in the load module for subsystem parameters) is too large to allocate to any available online disk device. This condition is normally accompanied by MVS ABEND message IEC032I.

  To solve the problem, make space available on the disk volumes, or make available another online storage volume for DB2. If this is not possible, an adjustment to the value of PRIQTY in the DSNZP*xxx* module is required to reduce the primary allocation. (For instructions, see Part 2 of *DB2 Installation Guide*. If the primary allocation is reduced, the size of the secondary space allocation might have to be increased to avoid future IEC030I abends.

## Temporary resource failure

DB2 sometimes experiences a temporary problem when it accesses log data sets. For example, DB2 might experience a problem when it attempts to allocate or open archive log data sets during the rollback of a long-running unit of recovery. These temporary failures can be caused by:

- A temporary problem with DFHSM recall
- A temporary problem with the tape subsystem
- Uncataloged archive logs
- Archive tape mount requests being cancelled

In these cases, DB2 issues messages for the access failure for each log data set. These messages provide information needed to resolve the access error. For example:

```
DSNJ104I  ( DSNJR206 RECEIVED ERROR STATUS 00000004
          FROM DSNPCLOC FOR DSNAME=DSNC710.ARCHLOG1.A0000049

DSNJ104I  ( DSNJR206 RECEIVED ERROR STATUS 00000004
          FROM DSNPCLOC FOR DSNAME=DSNC710.ARCHLOG2.A0000049

*DSNJ153E ( DSNJR006 CRITICAL LOG READ ERROR
          CONNECTION-ID = TEST0001
          CORRELATION-ID = CTHDCORID001
          LUWID = V71A.SYEC1DB2.B3943707629D=10
          REASON-CODE = 00D10345
```

You can attempt to recover from temporary failures by issuing a positive reply to message:

```
*26 DSNJ154I  ( DSNJR126 REPLY Y TO RETRY LOG READ REQUEST, N TO ABEND
```

If the problem persists, quiesce other work in the system before replying **N**, which terminates DB2.

## BSDS failure

For information about the BSDS, see “Managing the bootstrap data set (BSDS)” on page 341. Normally, there are two copies of the BSDS; but if one is damaged, DB2 immediately falls into single BSDS mode processing. The damaged copy of the BSDS must be recovered prior to the next restart. If you are in single mode and damage the only copy of the BSDS, or if you are in dual mode and damage both copies, DB2 stops until the BSDS is recovered. To proceed under these conditions, see “Recovering the BSDS from a backup copy” on page 431.

This section covers some of the BSDS problems that can occur. Problems not covered here include:

- RECOVER BSDS command failure (messages DSNJ301I through DSNJ307I)
- Change log inventory utility failure (message DSNJ123E)
- Errors in the BSDS backup being dumped by offload (message DSNJ125I).

See Part 2 of *DB2 Messages and Codes* for information about those problems.

## Problem 1 - An I/O error occurs

***Symptom:*** The following message appears:

```
DSNJ126I - BSDS ERROR FORCED SINGLE BSDS MODE
```

It is followed by one of these messages:

```
DSNJ107I - READ ERROR ON BSDS
           DSNAME=... ERROR STATUS=...

DSNJ108I - WRITE  ERROR ON BSDS
           DSNAME=... ERROR STATUS=...
```

***System action:*** The BSDS mode changes from dual to single.

***System programmer action:***

1. Use access method services to rename or delete the damaged BSDS and to define a new BSDS with the same name as the failing BSDS. Control statements can be found in job DSNTIJIN.
2. Issue the DB2 command RECOVER BSDS to make a copy of the good BSDS in the newly allocated data set and to reinstate dual BSDS mode.

## Problem 2 - An error occurs while opening

***Symptom:*** The following message appears:

```
DSNJ100I - ERROR OPENING BSDSn DSNAME=..., ERROR STATUS=eeii
```

The error status is VSAM return code/feedback. For information about VSAM codes, refer to *DFSMS/MVS: Macro Instructions for Data Sets*.

***System action:*** None.

***System programmer action:***

1. Use access method services to delete or rename the damaged data set, to define a replacement data set, and to copy the remaining BSDS to the replacement with the REPRO command.
2. Use the command START DB2 to start the DB2 subsystem.

## Problem 3 - Unequal timestamps exist

***Symptom:*** The following message appears:

```
DSNJ120I - DUAL BSDS DATA SETS HAVE UNEQUAL TIMESTAMPS,
           BSDS1 SYSTEM=..., UTILITY=..., BSDS2 SYSTEM=..., UTILITY=...
```

The following are possible causes:

- One of the volumes containing the BSDS has been restored. All information of the restored volume is down-level. If the volume contains any active log data sets or DB2 data, their contents are also down-level. The down-level volume has the lower timestamp.

  For information about resolving this problem, see “Failure during a log RBA read request” on page 493.
- Dual BSDS mode has degraded to single BSDS mode, and you are trying to start without recovering the bad BSDS.
- The DB2 subsystem abended after updating one copy of the BSDS, but prior to updating the second copy.

***System action:*** None.

***System programmer action:***

1. Run the print log map utility (DSNJU004) on both copies of the BSDS; compare the lists to determine which copy is accurate or current.
2. Rename the down-level data set and define a replacement for it.
3. Copy the good data set to the replacement data set, using the REPRO command of access method services.
4. If the problem was caused by a restored down-level BSDS volume, and:
   - if the restored volume contains active log data, and
   - you were using dual active logs on separate volumes

   then use access method services REPRO to copy the current version of the active log to the down-level data set.

   If you were not using dual active logs, you must cold start the subsystem. (For this procedure, see “Failure resulting from total or excessive loss of log data” on page 496 ).

   If the restored volume contains database data, use the RECOVER utility to recover that data after successful restart.

## Recovering the BSDS from a backup copy

If DB2 is operating in single BSDS mode and the BSDS is damaged, or if DB2 is operating in dual BSDS mode and both BSDSs are damaged, DB2 stops and does not restart until dual BSDS is restored. In this case, take the following steps:

1. Locate the BSDS associated with the most recent archive log data set. The data set name of the most recent archive log appears on the MVS console in the last occurrence of message DSNJ003I, which indicates that offloading has successfully completed. In preparation for the rest of this procedure, it is a good practice to keep a log of all successful archives noted by that message.
   - If archive logs are on disk, the BSDS is allocated on any available disk. The BSDS name is like the corresponding archive log data set name; change only the first letter of the last qualifier, from A to B, as in the example below:

     **Archive log name**
     DSN.ARCHLOG1.**A**0000001

     **BSDS copy name**
     DSN.ARCHLOG1.**B**0000001
   - If archive logs are on tape, the BSDS is the first data set of the first archive log volume. The BSDS is not repeated on later volumes.

2. If the most recent archive log data set has no copy of the BSDS (presumably because an error occurred when offloading it), then locate an earlier copy of the BSDS from an earlier offload.
3. Rename any *damaged* BSDS by using the access method services ALTER command with the NEWNAME option. If the decision is made to delete any damaged BSDS, use the access method services DELETE command. For each damaged BSDS, use access method services to define a new BSDS as a replacement data set. Job DSNTIJIN contains access method services control statements to define a new BSDS.

   The BSDS is a VSAM key-sequenced data set that has three components: cluster, index, and data. You must rename *all* components of the data set. Avoid changing the high-level qualifier. See *DFSMS/MVS: Access Method Services for VSAM Catalogs* for detailed information about using the access method services ALTER command.
4. Use the access method services REPRO command to copy the BSDS from the archive log to one of the replacement BSDSs you defined in step 3. Do not copy any data to the second replacement BSDS; data is placed in the second replacement BSDS in a later step in this procedure.
   a. Print the contents of the replacement BSDS.

      Use the print log map utility (DSNJU004) to print the contents of the replacement BSDS. This enables you to review the contents of the replacement BSDS before continuing your recovery work.
   b. Update the archive log data set inventory in the replacement BSDS.

      Examine the print log map output and note that the replacement BSDS does not obtain a record of the archive log from which the BSDS was copied. If the replacement BSDS is a particularly old copy, it is missing all archive log data sets that were created later than the BSDS backup copy. Thus, the BSDS inventory of the archive log data sets must be updated to reflect the current subsystem inventory.

      Use the change log inventory utility (DSNJU003) NEWLOG statement to update the replacement BSDS, adding a record of the archive log from which the BSDS was copied. Make certain the CATALOG option of the NEWLOG statement is properly set to CATALOG = YES if the archive log data set is cataloged. Also, use the NEWLOG statement to add any additional archive log data sets that were created later than the BSDS copy.
   c. Update DDF information in the replacement BSDS.

      If your installation’s DB2 is part of a distributed network, the BSDS contains the DDF control record. You must review the contents of this record in the output of the print log map utility. If changes are required, use the change log inventory DDF statement to update the BSDS DDF record.
   d. Update the active log data set inventory in the replacement BSDS.

      In unusual circumstances, your installation could have added, deleted, or renamed active log data sets since the BSDS was copied. In this case, the replacement BSDS does not reflect the actual number or names of the active log data sets your installation has currently in use.

      If you must delete an active log data set from the replacement BSDS log inventory, use the change log inventory utility DELETE statement.

      If you need to add an active log data set to the replacement BSDS log inventory, use the change log inventory utility NEWLOG statement. Be certain that the RBA range is specified correctly on the NEWLOG statement.

If you must rename an active log data set in the replacement BSDS log inventory, use the change log inventory utility DELETE statement, followed by the NEWLOG statement. Be certain that the RBA range is specified correctly on the NEWLOG statement.

e. Update the active log RBA ranges in the replacement BSDS.

   Later, when a restart is performed, DB2 compares the RBAs of the active log data sets listed in the BSDS with the RBAs found in the actual active log data sets. If the RBAs do not agree, DB2 does not restart. The problem is magnified when a particularly old copy of the BSDS is used. To resolve this problem, you can use the change log inventory utility to adjust the RBAs found in the BSDS with the RBAs in the actual active log data sets. This can be accomplished by the following:

   - If you are not certain of the RBA range of a particular active log data set, use DSN1LOGP to print the contents of the active log data set. Obtain the logical starting and ending RBA values for the active log data set from the DSN1LOGP output. The STARTRBA value you use in the change log inventory utility must be at the beginning of a control interval. Similarly, the ENDRBA value you use must be at the end of a control interval. To get these values, round the starting RBA value from the DSN1LOGP output down so that it ends in X'000'. Round the ending RBA value up so that it ends in X'FFF'.
   - When the RBAs of all active log data sets are known, compare the actual RBA ranges with the RBA ranges found in the BSDS (listed in the print log map utility output).

     If the RBA ranges are equal for all active log data sets, you can proceed to the next recovery step without any additional work.

     If the RBA ranges are not equal, then the values in the BSDS must be adjusted to reflect the actual values. For each active log data set that needs to have the RBA range adjusted, use the change log inventory utility DELETE statement to delete the active log data set from the inventory in the replacement BSDS. Then use the NEWLOG statement to redefine the active log data set to the BSDS.

f. If only two active log data sets are specified in the replacement BSDS, add a new active log data set for each copy of the active log and define each new active log data set of the replacement BSDS log inventory.

   If only two active log data sets are specified for each copy of the active log, DB2 can have difficulty during restart. The difficulty can arise when one of the active log data sets is full and has not been offloaded, while the second active log data set is close to filling. Adding a new active log data set for each copy of the active log can alleviate difficulties on restart in this scenario.

   To add a new active log data set for each copy of the active log, use the access method services DEFINE command to define a new active log data set for each copy of the active log. The control statements to accomplish this task can be found in job DSNTIJIN. Once the active log data sets are physically defined and allocated, use the change log inventory utility NEWLOG statement to define the new active log data sets of the replacement BSDS. The RBA ranges need not be specified on the NEWLOG statement.

5. Copy the updated BSDS copy to the second new BSDS data set. The dual bootstrap data sets are now identical.

   You should consider using the print log map utility (DSNJU004) to print the contents of the second replacement BSDS at this point.

6. See “Chapter 23. Recovery from BSDS or log failure during restart” on page 475 for information about what to do if you have lost your current active log data set. For a discussion of how to construct a conditional restart record, see “Step 4: Truncate the log at the point of error” on page 485.
7. Restart DB2, using the newly constructed BSDS. DB2 determines the current RBA and what active logs need to be archived.

## DB2 database failures

***Problem:*** Allocation or open problems occur.

***Symptom 1:*** The following message indicates an allocation problem:

```
DSNB207I - DYNAMIC ALLOCATION OF DATA SET FAILED.
           REASON=rrrr DSNAME=dsn
```

where *rrrr* is an MVS dynamic allocation reason code. For information about these reason codes, see *OS/390 MVS Programming: Authorized Assembler Services Guide*.

***Symptom 2:*** The following messages indicate a problem at open:

```
IEC161I  rc[(sfi)] - ccc, iii,  sss, ddn,
         ddd, ser, xxx, dsn, cat
```

where:

| | |
|---|---|
| *rc* | Is a return code |
| *sfi* | Is subfunction information (*sfi* only appears with certain return codes) |
| *ccc* | Is a function code |
| *iii* | Is a job name |
| *sss* | Is a step name |
| *ddn* | Is a *ddname* |
| *ddd* | Is a device number (if the error is related to a specific device) |
| *ser* | Is a volume serial number (if the error is related to a specific volume) |
| *xxx* | Is a VSAM cluster name |
| *dsn* | Is a data set name |
| *cat* | Is a catalog name. |

For information about these codes, see *OS/390 MVS System Messages Volume 1*.

```
DSNB204I - OPEN OF DATA SET FAILED. DSNAME = dsn
```

***System action:***
- The table space is automatically stopped.
- Programs receive an -904 SQLCODE (SQLSTATE '57011').
- If the problem occurs during restart, the table space is marked for deferred restart, and restart continues. The changes are applied later when the table space is started.

***System programmer action:*** None.

***Operator action:***
1. Check reason codes and correct.
2. Ensure that drives are available for allocation.
3. Enter the command START DATABASE.

## Recovery from down-level page sets

When using a stand-alone or non-DB2 utility, such as DSN1COPY or DFSMShsm, it is possible to replace a DB2 data set by mistake with an incorrect or outdated copy. Such a copy is called *down-level*; using it can cause complex problems.

Other reasons for a down-level condition are:

- A cold start of DB2 was performed.
- The VSAM high-used RBA of a table space has become corrupted.

DB2 associates a level ID with every page set or partition. Most operations detect a down-level ID, and return an error condition, when the page set or partition is first opened for mainline or restart processing. The exceptions are operations that do not use the data:

LOAD REPLACE
RECOVER
REBUILD INDEX
DSN1COPY
DSN1PRNT

The RESET option of DSN1COPY resets the level ID on its output to a neutral value that passes any level check. Hence, you can still use DSN1COPY to move data from one system or table space to another.

The LEVELID option of the REPAIR utility marks a down-level table space or index as current. See Part 2 of *DB2 Utility Guide and Reference* for details on using REPAIR.

For directory page sets, and the page sets for SYSIBM.SYSCOPY and SYSIBM.SYSGROUP, a down-level ID is detected only at restart and not during mainline operations.

***Symptom:*** The following message appears:

```
DSNB232I csect-name - UNEXPECTED DATA SET LEVEL ID ENCOUNTERED
```

The message contains also the level ID of the data set, the level ID that DB2 expects, and the name of the data set.

***System action:***

- If the error was reported during mainline processing, DB2 sends back a ″resource unavailable″ SQLCODE to the application and a reason code explaining the error.
- If the error was detected while a utility was processing, the utility gives a return code 8.

***System programmer action:*** You can recover in any of the following ways:

If the message occurs during restart:

- Replace the data set with one at the proper level, using DSN1COPY, DFSMShsm, or some equivalent method. To check the level ID of the new data set, run the stand-alone utility DSN1PRNT on it, with the options PRINT(0) (to print only the header page) and FORMAT. The formatted print identifies the level ID.

- Recover the data set to the current time, or to a prior time, using the RECOVER utility.
- Replace the contents of the data set, using LOAD REPLACE.

If the message occurs during normal operation, use any of the methods listed above, plus one more:

- Accept the down-level data set by changing its level ID.

  The REPAIR utility contains a statement for that purpose. Run a utility job with the statement REPAIR LEVELID. The LEVELID statement cannot be used in the same job step with any other REPAIR statement.

**Important**

If you accept a down-level data set or disable down-level detection, your data might be inconsistent.

For more information about using the utilities, see *DB2 Utility Guide and Reference*.

You can control down-level detection. Use the LEVELID UPDATE FREQ field of panel DSNTIPL to either disable down-level detection or control how often the level ID of a page set or partition is updated. DB2 accepts any value between 0 and 32767.

To disable down-level detection, specify 0 in the LEVELID UPDATE FREQ field of panel DSNTIPL.

To control how often level ID updates are taken, specify a value between 1 and 32767. See Part 2 of *DB2 Installation Guide* for more information about choosing the frequency of level ID updates.

## Procedure for recovering invalid LOBs

Unless your LOBs are fairly small, specifying LOG NO for LOB objects is recommended. The performance cost of logging exceeds the benefits you can receive from logging such large amounts of data. If no changes are made to LOB data, this is not an issue. However, you should make image copies of the LOB table space to prepare for failures. The frequency with which you make image copies is based on how often you update LOB data.

If you need to recover LOB data that changed after your last image copy, follow this procedure:

1. Run the RECOVER utility as you do for other table spaces:

   ```
   RECOVER TABLESPACE dbname.lobts
   ```

   If changes were made after the image copy, DB2 puts the table space in *Aux Warning* status. The purpose of this status is let you know that some of your LOBs are invalid. Applications that try to retrieve the values of those LOBs will receive SQLCODE -904. Applications can still access other LOBs in the LOB table space.
2. Get a report of the invalid LOBs by running CHECK LOB on the LOB table space:

   ```
   CHECK LOB TABLESPACE dbname.lobts
   ```

DB2 generates messages like the following one:

```
LOB WITH ROWID = 'xxxxxxx' VERSION = n  IS INVALID
```

3. Fix the invalid LOBs, by updating the LOBs or setting them to the null value. For example, suppose you determine from the CHECK LOB utility that the row of the EMP_PHOTO_RESUME table with ROWID X'C1BDC4652940D40A81C201AA0A28' has an invalid value for column RESUME. If host variable hvlob contains the correct value for RESUME, you can use this statement to correct the value:

```
UPDATE DSN8710.EMP_PHOTO_RESUME
  SET RESUME = :hvlob
  WHERE EMP_ROWID = ROWID(X'C1BDC4652940D40A81C201AA0A28');
```

## Table space input/output errors

***Problem:*** A table space failed.

***Symptom:*** The following message is issued:

```
DSNU086I  DSNUCDA1 READ I/O ERRORS ON SPACE= dddddddd.
          DATA SET NUMBER= nnn.
          I/O ERROR PAGE RANGE= aaaaaa, bbbbbb.
```

where *dddddddd* is a table space name.

Any table spaces identified in DSNU086I messages must be recovered using one of the procedures in this section listed under “Operator Action”.

***System action:*** DB2 remains active.

***Operator action:*** Fix the error range.

1. Use the command STOP DATABASE to stop the failing table space.
2. Use the command START DATABASE ACCESS (UT) to start the table space for utility-only access.
3. Start a RECOVER utility step to recover the error range by using the DB2 RECOVER (*dddddddd*) ERROR RANGE statement.

   If you receive message DSNU086I again, indicating the error range recovery cannot be performed, use the recovery procedure below.
4. Give the command START DATABASE to start the table space for RO or RW access, whichever is appropriate. If the table space is recovered, you do not need to continue with the procedure below.

***If error range recovery fails:*** If the error range recovery of the table space failed because of a hardware problem, proceed as follows:

1. Use the command STOP DATABASE to stop the table space or table space partition that contains the error range. This causes all the in-storage data buffers associated with the data set to be externalized to ensure data consistency during the subsequent steps.
2. Use the INSPECT function of the IBM Device Support Facility, ICKDSF, to check for track defects and to assign alternate tracks as necessary. The physical location of the defects can be determined by analyzing the output of messages DSNB224I, DSNU086I, IOS000I, which were displayed on the system operator’s console at the time the error range was created. If damaged storage media is suspected, then request assistance from hardware support personnel before proceeding. Refer to *Device Support Facilities User's Guide and Reference* for information about using ICKDSF.

3. Use the command START DATABASE to start the table space with ACCESS(UT) or ACCESS(RW).
4. Run the utility RECOVER ERROR RANGE that, from image copies, locates, allocates, and applies the pages within the tracks affected by the error ranges.

## DB2 catalog or directory input/output errors

***Problem:*** The DB2 catalog or directory failed.

***Symptom:*** The following message is issued:

```
DSNU086I  DSNUCDA1 READ I/O ERRORS ON SPACE= dddddddd.
          DATA SET NUMBER= nnn.
          I/O ERROR PAGE RANGE= aaaaaa, bbbbbb.
```

where *dddddddd* is a table space name from the catalog or directory. *dddddddd* is the table space that failed (for example, SYSCOPY, abbreviation for SYSIBM.SYSCOPY, or SYSLGRNX, abbreviation for DSNDB01.SYSLGRNX). This message can indicate either read or write errors. You can also get a DSNB224I or DSNB225I message, which could indicate an input or output error for the catalog or directory.

Any catalog or directory table spaces that are identified in DSNU086I messages must be recovered with this procedure.

***System action:*** DB2 remains active.

If the DB2 directory or any catalog table is damaged, only user IDs with the RECOVERDB privilege in DSNDB06, or an authority that includes that privilege, can do the recovery. Furthermore, until the recovery takes place, only those IDs can do anything with the subsystem. If an ID without proper authorization attempts to recover the catalog or directory, message DSNU060I is displayed. If the authorization tables are unavailable, message DSNT500I is displayed indicating the resource is unavailable.

***System programmer action:*** None.

***Operator action:*** Take the following steps for each table space in the DB2 catalog and directory that has failed. If there is more than one, refer to the description of RECOVER in Part 2 of *DB2 Utility Guide and Reference* for more information about the specific order of recovery.

1. Stop the failing table spaces.
2. Determine the name of the data set that failed. There are two ways to do this:
   - Check *prefix*.SDSNSAMP (DSNTIJIN), which contains the JCL for installing DB2. Find the fully qualified name of the data set that failed by searching for the name of the table space that failed (the one identified in the message as SPACE = *dddddddd*).
   - Construct the data set name by doing one of the following:
     – If the table space is in the DB2 catalog, the data set name format is:

       ```
       DSNC710.DSNDBC.DSNDB06.dddddddd.I0001.A001
       ```

       where *dddddddd* is the name of the table space that failed.
     – If the table space is in the DB2 directory, the data set name format is:

       ```
       DSNC710.DSNDBC.DSNDB01.dddddddd.I0001.A001
       ```

where *dddddddd* is the name of the table space that failed.

If you do not use the default (IBM-supplied) formats, the formats for data set names can be different.

3. Use access method services DELETE to delete the data set, specifying the fully qualified data set name.
4. After the data set has been deleted, use access method services DEFINE to redefine the same data set, again specifying the same fully qualified data set name. Use the JCL for installing DB2 to determine the appropriate parameters.

   **Important:** The REUSE parameter must be coded in the DEFINE statements.
5. Give the command START DATABASE ACCESS(UT), naming the table space involved.
6. Use the RECOVER utility to recover the table space that failed.
7. Give the command START DATABASE, specifying the table space name and RO or RW access, whichever is appropriate.

# Integrated catalog facility catalog VSAM volume data set failures

This section includes information regarding volume data set failures. The following topics are described:

"VSAM volume data set (VVDS) destroyed"
"Out of disk space or extent limit reached" on page 440

## VSAM volume data set (VVDS) destroyed

***Problem:*** A VSAM volume data set (VVDS) is either out of space or destroyed.

***Symptom:*** DB2 sends the following message to the master console.

```
DSNP012I - DSNPSCT0 - ERROR IN VSAM CATALOG LOCATE FUNCTION
                 FOR data_set_name
                 CTLGRC=50
                 CTLGRSN=zzzzRRRR
                 CONNECTION-ID=xxxxxxxx,
                 CORRELATION-ID=yyyyyyyyyyyy
                 LUW-ID=logical-unit-of-work-id=token
```

For a detailed explanation of this message, see Part 2 of *DB2 Messages and Codes*.

VSAM can also issue the following message:

```
IDC3009I VSAM CATALOG RETURN CODE IS 50, REASON CODE IS
         IGGOCLaa - yy
```

In this VSAM message, *yy* is 28, 30, or 32 for an out-of-space condition. Any other values for *yy* indicate a damaged VVDS.

***System action:*** Your program is terminated abnormally and one or more messages are issued.

***System programmer action:*** None.

***Operator action***: For information on recovering the VVDS, consult the appropriate book for the level of DFSMS/MVS you are using:

*DFSMS/MVS: Access Method Services for the Integrated Catalog*
*DFSMS/MVS: Managing Catalogs*

The procedures given in these books describe three basic recovery scenarios. First determine which scenario exists for the specific VVDS in error. Then, before beginning the appropriate procedure, take the following steps:

1. Determine the names of all table spaces residing on the same volume as the VVDS. To determine the table space names, look at the VTOC entries list for that volume, which indicates the names of all the data sets on that volume. For information on how to determine the table space name from the data set name, refer to "Part 2. Designing a database: advanced topics" on page 27.
2. Use the DB2 COPY utility to take image copies of all table spaces of the volume. Taking image copies minimizes reliance on the DB2 recovery log and can speed up the processing of the DB2 RECOVER utility (to be mentioned in a subsequent step).

   If the COPY utility cannot be used, continue with this procedure. Be aware that processing time increases because more information is obtained from the DB2 recovery log.
3. Use the command STOP DATABASE for all the table spaces that reside on the volume, or use the command STOP DB2 to stop the entire DB2 subsystem if an unusually large number or critical set of table spaces are involved.
4. If possible, use access method services to export all non-DB2 data sets residing on that volume. For more information, see *DFSMS/MVS: Access Method Services for the Integrated Catalog* and *DFSMS/MVS: Managing Catalogs*.
5. To recover all non-DB2 data sets on the volume, see *DFSMS/MVS: Access Method Services for the Integrated Catalog* and *DFSMS/MVS: Managing Catalogs*.
6. Use access method services DELETE and DEFINE commands to delete and redefine the data sets for all user-defined table spaces and DB2-defined data sets when the physical data set has been destroyed. DB2 automatically deletes and redefines all other STOGROUP defined table spaces.

   You do not need to do this for those table spaces that are STOGROUP defined; DB2 takes care of them automatically.
7. Issue the DB2 START DATABASE command to restart all the table spaces stopped in step 3. If the entire DB2 subsystem was stopped, issue the -START DB2 command.
8. Use the DB2 RECOVER utility to recover any table spaces and indexes. For information on recovering table spaces, refer to "Chapter 21. Backing up and recovering databases" on page 373.

## Out of disk space or extent limit reached

***Problem:*** There is no more space on the volume on which the data set is stored or the data set might have reached its maximum DB2 size or its maximum number of VSAM extents.

***Symptom:*** One of the following messages:

1. Extend request failure

   When an insert or update requires additional space, but the space is not available in the current table or index space, DB2 issues the following message:

   ```
   DSNP007I - DSNPmmmm - EXTEND FAILED FOR
              data-set-name.  RC=rrrrrrrr
              CONNECTION-ID=xxxxxxxx,
              CORRELATION-ID=yyyyyyyyyyyy
              LUWID-ID=logical-unit-of-work-id=token
   ```
2. Look ahead warning

A look ahead warning occurs when there is enough space for a few inserts and updates, but the index space or table space is almost full. On an insert or update at the end of a page set, DB2 determines whether the data set has enough available space. DB2 uses the following values in this space calculation:

- The primary space quantity from the integrated catalog facility (ICF) catalog
- The secondary space quantity from the ICF catalog
- The allocation unit size

If there is not enough space, DB2 tries to extend the data set. If the extend request fails, then DB2 issues the following message:

```
DSNP001I - DSNPmmmm - data-set-name IS WITHIN
           nK BYTES OF AVAILABLE SPACE.
           RC=rrrrrrrr
           CONNECTION-ID=xxxxxxxx,
           CORRELATION-ID=yyyyyyyyyyyy
           LUW-ID=logical-unit-of-work-id=token
```

***System action:*** For a demand request failure during restart, the object supported by the data set (an index space or a table space) is stopped with deferred restart pending. Otherwise, the state of the object remains unchanged. Programs receive a -904 SQL return code (SQLSTATE '57011').

***System programmer action:*** None.

***Operator action:*** The appropriate choice of action depends on particular circumstances. The following topics are described in this section; decision criteria are outlined below:

- “Procedure 1. Extend a data set” on page 442
- “Procedure 2. Enlarge a fully extended data set (user-managed)” on page 442
- “Procedure 3. Enlarge a fully extended data set (in a DB2 storage group)” on page 442
- “Procedure 4. Add a data set” on page 443
- “Procedure 5. Redefine a partition” on page 443
- “Procedure 6. Enlarge a fully extended data set for the work file database” on page 443

If the database qualifier of the data set name is DSNDB07, then the condition is on your work file database. Use “Procedure 6. Enlarge a fully extended data set for the work file database” on page 443.

In all other cases, if the data set has *not* reached its maximum DB2 size, then you can enlarge it. (The maximum size is 2 gigabytes for a data set of a simple space, and 1, 2, or 4 gigabytes for a data set containing a partition. Large partitioned table spaces and indexes on large partitioned table spaces have a maximum data set size of 4 gigabytes.)

- If the data set has *not* reached the maximum number of VSAM extents, use “Procedure 1. Extend a data set” on page 442.
- If the data set *has* reached the maximum number of VSAM extents, use either “Procedure 2. Enlarge a fully extended data set (user-managed)” on page 442 or “Procedure 3. Enlarge a fully extended data set (in a DB2 storage group)” on page 442, depending on whether the data set is user-managed or DB2-managed. User-managed data sets include essential data sets such as the catalog and the directory.

If the data set *has* reached its maximum DB2 size, then your action depends on the type of object it supports.

- If the object is a simple space, add a data set, using “Procedure 4. Add a data set” on page 443.
- If the object is partitioned, each partition is restricted to a single data set. You must redefine the partitions; use “Procedure 5. Redefine a partition” on page 443.

**Procedure 1. Extend a data set:** If the data set is user-defined, provide more VSAM space. You can add volumes with the access method services command ALTER ADDVOLUMES or make room on the current volume.

If the data set is defined in a DB2 storage group, add more volumes to the storage group by using the SQL ALTER STOGROUP statement.

For more information on DB2 data set extension, refer to “Extending DB2-managed data sets” on page 39.

**Procedure 2. Enlarge a fully extended data set (user-managed):**

1. To allow for recovery in case of failure during this procedure, be sure that you have a recent full image copy (for table spaces or if you copy your indexes). Use the DSNUM option to identify the data set for table spaces or partitioning indexes.
2. Issue the command STOP DATABASE SPACENAM for the last data set of the object supported.
3. Delete the last data set by using access method services. Then redefine it and enlarge it as necessary.
4. Issue the command START DATABASE ACCESS (UT) to start the object for utility-only access.

   The object must be user-defined and a linear data set, and should not have reached the maximum number of 32 data sets (or 254 data sets for LOB table spaces). For non-partitioning indexes on a large partitioned table space, the maximum is 128 data sets.
5. To recover the data set that was redefined, use RECOVER on the table space or index, and identify the data set by the DSNUM option (specify this DSNUM option for table spaces or partitioning indexes only).

   RECOVER lets you specify a single data set number for a table space. Thus, only the last data set (the one that needs extension) must be redefined and recovered. This can be better than using REORG if the table space is very large and contains multiple data sets, and if the extension must be done quickly.

   If you do not copy your indexes, then use the REBUILD INDEX utility.
6. Issue the command START DATABASE to start the object for either RO or RW access, whichever is appropriate.

**Procedure 3. Enlarge a fully extended data set (in a DB2 storage group):**

1. Use ALTER TABLESPACE or ALTER INDEX with a USING clause. (You do not have to stop the table space before you use ALTER TABLESPACE.) You can give new values of PRIQTY and SECQTY in either the same or a new DB2 storage group.
2. Use one of the following procedures. Keep in mind that no movement of data occurs until this step is completed.
   - For indexes:

If you have taken full image copies of the index, run the RECOVER INDEX utility. Otherwise, run the REBUILD INDEX utility.

- For table spaces other than LOB table space:

  Run one of the following utilities on the table space: REORG, RECOVER, or LOAD REPLACE.
- For LOB table spaces defined with LOG YES:

  Run the RECOVER utility on the table space.
- For LOB table spaces defined with LOG NO, follow these steps:
  a. Start the table space in read-only (RO) mode to ensure that no updates are made during this process.
  b. Make an image copy of the table space.
  c. Run the RECOVER utility on the table space.
  d. Start the table space in read-write (RW) mode.

**Procedure 4. Add a data set:** If the object supported is user-defined, use the access method services to define another data set. The name of the new data set must continue the sequence begun by the names of the existing data sets that support the object. The last four characters of each name are a relative data set number: If the last name ended with A001, the next must end with A002, and so on. Also, be sure to add either **I** or **J** in the name of the data set.

If the object is defined in a DB2 storage group, DB2 automatically tries to create an additional data set. If that fails, access method services messages are sent to an operator indicating the cause of the problem. Correcting that problem allows DB2 to get the additional space.

**Procedure 5. Redefine a partition:**

1. Alter the key range values of the partitioning index.
2. Use REORG with inline statistics on the partitions that are affected by the change in key range.
3. Use RUNSTATS on the nonpartitioned indexes.
4. Rebind the dependent packages and plans.

**Procedure 6. Enlarge a fully extended data set for the work file database**

Use one of the following methods to add extension space to the storage group:

- Use SQL to create more table spaces in database DSNDB07.

Or,

- Execute these steps:
  1. Use the command STOP DATABASE(DSNDB07) to ensure that no users are accessing the database.
  2. Use SQL to alter the storage group, adding volumes as necessary.
  3. Use the command START DATABASE(DSNDB07) to allow access to the database.

## Violations of referential constraints

***Problem:*** A table space can contain violations of referential constraints.

***Symptom:*** One of the following messages is issued at the end of utility processing, depending upon whether or not the table space is partitioned.

```
DSNU561I csect-name - TABLESPACE= tablespace-name PARTITION= partnum
         IS IN CHECK PENDING
DSNU563I csect-name - TABLESPACE= tablespace-name IS IN CHECK PENDING
```

***System action:*** None. The table space is still available; however, it is not available to the COPY, REORG, and QUIESCE utilities, or to SQL select, insert, delete, or update operations that involve tables in the table space.

***System programmer action:*** None.

***Operator action:***

1. Use the START DATABASE ACCESS (UT) command to start the table space for utility-only access.
2. Run the CHECK DATA utility on the table space. Take the following into consideration:
   - If you do not believe that violations exist, specify DELETE NO. If, indeed, violations do not exist, this resets the check-pending status; however, if violations do exist, the status is not going to be reset.
   - If you believe that violations exist, specify the DELETE YES option and an appropriate exception table (see Part 2 of *DB2 Utility Guide and Reference*for the syntax of this utility). This deletes all rows in violation, copies them to an exception table, and resets the check-pending status.
   - If the check-pending status was set during execution of the LOAD utility, specify the SCOPE PENDING option. This checks only those rows added to the table space by LOAD, rather than every row in the table space.
3. Correct the rows in the exception table, if necessary, and use the SQL INSERT statement to insert them into the original table.
4. Give the command START DATABASE to start the table space for RO or RW access, whichever is appropriate. The table space is no longer in check-pending status and is available for use. If you use the ACCESS (FORCE) option of this command, the check-pending status is reset. However, this is not recommended because it does not correct violations of referential constraints.

# Failures related to the distributed data facility

The following failures related to the DDF are discussed in this section:

## Conversation failure

***Problem:*** A VTAM APPC or TCP/IP conversation failed during or after allocation and is unavailable for use.

***Symptom:*** VTAM or TCP/IP returns a resource unavailable condition along with the appropriate diagnostic reason code and message. A DSNL500 or DSNL511 (conversation failed) message is sent to the console for the first failure to a location for a specific logical unit (LU) mode or TCP/IP address. All other threads detecting a failure from that LU mode or IP address are suppressed until communications to that LU using that mode are successful.

DB2 returns messages DSNL501I and DSNL502I. Message DSNL501I usually means that the other subsystem is not up.

***System action:*** When the error is detected, it is reported by a console message and the application receives an SQL return code. For DB2 private protocol access, SQLCODE -904 (SQLSTATE '57011') is returned with resource type 1001, 1002, or 1003. The resource name in the SQLCA contains VTAM return codes such as RTNCD, FDBK2, RCPRI, and RCSEC, and any SNA SENSE information. See *VTAM for MVS/ESA Messages and Codes* for more information.

If you use application directed access or DRDA as the database protocols, SQLCODE -30080 is returned to the application. The SQLCA contains the VTAM diagnostic information, which contains only the RCPRI and RCSEC codes. For SNA communications errors, SQLCODE -30080 is returned. For TCP/IP connections, SQLCODE -30081 is returned. See *DB2 Messages and Codes* for more information about those SQL return codes.

The application can choose to request rollback or commit. Commit or rollback processing deallocates all but the first conversation between the allied thread and the remote database access thread. A commit or rollback message is sent over this remaining conversation.

Errors during the conversation’s deallocation process are reported through messages, but do not stop the commit or rollback processing. If the conversation used for the commit or roll back message fails, the error is reported. If the error occurred during a commit process, the commit process continues, provided the remote database access was read only; otherwise the commit process is rolled back.

***System programmer action:*** The system programmer needs to review the VTAM or TCP/IP return codes and might need to discuss the problem with a communications expert. Many VTAM or TCP/IP errors, besides the error of an inactive remote LU or TCP/IP errors, require a person who has a knowledge of VTAM or TCP/IP and the network configuration to diagnose them.

***Operator action:*** Correct the cause of the unavailable resource condition by taking action required by the diagnostic messages appearing on the console.

## Communications database failure

This section describes two different problems.

### Problem 1

A failure occurs during an attempt to access the DB2 CDB (after DDF is started).

***Symptom:*** A DSNL700I message, indicating that a resource unavailable condition exists, is sent to the console. Other messages describing the cause of the failure are also sent to the console.

***System action:*** The distributed data facility (DDF) does not terminate if it has already started and an individual CDB table becomes unavailable. Depending on the severity of the failure, threads will either receive a -904 SQL return code (SQLSTATE '57011') with resource type 1004 (CDB), or continue using VTAM defaults. Only the threads that access locations that have not had any prior threads will receive a -904 SQL return code. DB2 and DDF remain up.

***Operator action:*** Correct the error based on the messages received, then stop and restart DDF.

### Problem 2

The DB2 CDB is not defined correctly. This occurs when DDF is started and the DB2 catalog is accessed to verify the CDB definitions.

***Symptom:*** A DSNL701I, 702I, 703I, 704I, or 705I message is issued to identify the problem. Other messages describing the cause of the failure are also sent to the console.

***System action:*** DDF fails to start up. DB2 remains up.

***Operator action:*** Correct the error based on the messages received and restart DDF.

## Failure of a database access thread

***Problem:*** A database access thread has been deallocated and a conversation failure occurs.

***Symptom:*** In the event of a failure of a database access thread, the DB2 server terminates the database access thread only if a unit of recovery exists. The server deallocates the database access thread and then deallocates the conversation with an abnormal indication (a negative SQL code), which is subsequently returned to the requesting application. The returned SQL code depends on the type of remote access:

- DB2 private protocol access

  The application program receives a -904 SQL return code (SQLSTATE '57011') with a resource type 1005 at the requesting site. The SNA sense in the resource name contains the DB2 reason code describing the failure.

- DRDA access

  For a database access thread or non-DB2 server, a DDM error message is sent to the requesting site and the conversation is deallocated normally. The SQL error status code is a -30020 with a resource type ’1232’ (agent permanent error received from the server).

***System action:*** Normal DB2 error recovery mechanisms apply with the following exceptions:

- Errors caught in the functional recovery routine are automatically converted to rollback situations. The allied thread sees conversation failures.
- Errors occurring during commit, roll back, and deallocate *within the DDF function* do not normally cause DB2 to abend. Conversations are deallocated and the database access thread is terminated. The allied thread sees conversation failures.

***System programmer action:*** All diagnostic information related to the failure must be collected at the serving site. For a DB2 DBAT, a dump is produced at the server.

***Operator action:*** Communicate with the operator at the other site to take the appropriate corrective action, regarding the messages appearing on consoles at both the requesting and responding sites. Operators at both sites should gather the appropriate diagnostic information and give it to the programmer for diagnosis.

## VTAM failure

***Problem:*** VTAM terminates or fails.

***Symptom:*** VTAM messages and DB2 messages are issued indicating that DDF is terminating and explaining why.

***System action:*** DDF terminates.

An abnormal VTAM failure or termination causes DDF to issue a STOP DDF MODE(FORCE) command. The VTAM commands Z NET,QUICK or Z NET,CANCEL causes an abnormal VTAM termination. A Z NET,HALT causes a -STOP DDF MODE(QUIESCE) to be issued by DDF.

***System programmer action:*** None.

***Operator action:*** Correct the condition described in the messages received at the console, and restart VTAM and DDF.

## TCP/IP failure

***Problem:*** TCP/IP terminates or fails.

***Symptom:*** TCP/IP messages and DB2 messages are issued indicating that TCP/IP is unavailable.

***System action:*** DDF periodically attempts to reconnect to TCP/IP. If the TCP/IP listener fails, DDF automatically tries to reestablish the TCP/IP listener for the SQL port or the resync port every 3 minutes. TCP/IP connections cannot be established until the TCP/IP listener is reestablished.

***System programmer action:*** None.

***Operator action:*** Correct the condition described in the messages received at the console, restart TCP/IP. You do not have to restart DDF after a TCP/IP failure.

## Failure of a remote logical unit

***Problem:*** A series of conversation or change number of sessions (CNOS) failures occur from a remote LU.

***Symptom:*** Message DSNL501I is issued when a CNOS request to a remote LU fails. The CNOS request is the first attempt to connect to the remote site and must be negotiated before any conversations can be allocated. Consequently, if the remote LU is not active, message DSNL500I is displayed indicating that the CNOS request cannot be negotiated. Message DSNL500I is issued only once for all the SQL conversations that fail because of a remote LU failure.

Message DSNL502I is issued for system conversations that are active to the remote LU at the time of the failure. This message contains the VTAM diagnostic information on the cause of the failure.

***System action:*** Any application communications with a failed LU receives a message indicating a resource unavailable condition. The application programs receive SQL return code -904 (SQLSTATE '57011') for DB2 private protocol access and SQL return code -30080 for DRDA access. Any attempt to establish communication with such an LU fails.

***Operator action:*** Communicate with the other sites involved regarding the unavailable resource condition, and request that appropriate corrective action be taken. If a DSNL502 message is received, the operator should activate the remote LU.

## Indefinite wait conditions for distributed threads

***Problem:*** An allied thread is waiting indefinitely for a response from a remote subsystem or a database access thread is waiting for a response from the local subsystem.

***Symptom:*** An application is in an indefinitely long wait condition. This can cause other DB2 threads to fail due to resources held by the waiting thread. DB2 sends an error message to the console and the application program receives an SQL return code.

***System action:*** None.

***System programmer action:*** None.

***Operator action:*** Use the DISPLAY THREAD command with the LOCATION and DETAIL options to identify the LUWID and the session's allocation for the waiting thread. Then use the CANCEL DDF THREAD command to cancel the waiting thread. If the CANCEL DDF THREAD command fails to break the wait (because the thread is not suspended in DB2), try using VTAM commands such as VARY TERM,SID=xxx. For additional information concerning canceling DDF threads, see "The command CANCEL THREAD" on page 317 and "Using VTAM commands to cancel threads" on page 319.

To check for very long waits, look to see if the conversation timestamp is changing from the last time used. If it is changing, the conversation thread is not hung, but is taking more time for a long query. Also, look for conversation state changes and determine what they mean.

## Security failures for database access threads

***Problem:*** During database access thread allocation, the remote user does not have the proper security to access DB2 through the DDF.

***Symptom:*** Message DSNL500I is issued at the requester for VTAM conversations (if it is a DB2 subsystem) with return codes RTNCD=0, FDBK2=B, RCPRI=4, RCSEC=5 meaning ″Security Not Valid.″ The server has deallocated the conversation because the user is not allowed to access the server. For conversations using DRDA access, LU 6.2 communications protocols present specific reasons for why the user failed, to be returned to the application. If the server is a DB2 database access thread, message DSNL030I is issued to describe what caused the user to be denied access into DB2 through DDF. No message is issued for TCP/IP connections.

***System action:*** If the server is a DB2 subsystem, message DSNL030I is issued. Otherwise, the system programmer needs to refer to the documentation of the server. If the application uses DB2 private protocol access, it receives SQLCODE -904 (SQLSTATE '57011') with a reason code 00D3103D, indicating that a resource is unavailable. For DRDA access, SQLCODE –30082 is returned. See *DB2 Messages and Codes* for more information about those messages.

***System programmer action:*** Refer to the description of 00D3103D in Part 3 of *DB2 Messages and Codes*.

***Operator action:*** If it is a DB2 database access thread, the operator should provide the DSNL030I message to the system programmer. If it is not a DB2 server, the operator needs to work with the operator or programmer at the server to get diagnostic information needed by the system programmer.

## Remote site recovery from disaster at a local site

The procedures in this scenario differ from other recovery procedures in that the hardware at your local DB2 site cannot be used to recover data. This scenario bases recovery on the latest available archive log and assumes that all copies and reports have arrived at the recovery site as specified in “Preparing for disaster recovery” on page 385. For data sharing, see Chapter 5 of *DB2 Data Sharing: Planning and Administration* for the data sharing specific disaster recovery procedures.

***Problem:*** Your local system experiences damage or disruption that prevents recovery from that site.

***Symptom:*** Your local system hardware has suffered physical damage and is inoperable.

***System programmer action:*** Coordinate activities detailed below.

***Operator action (at the recovery site):***

| 1. For scenarios other than data sharing, continue with the next step.

**Data sharing**

Clean out old information from the coupling facility, if you have information in your coupling facility from practice startups. If you do not have old information in the coupling facility, you can omit this step.

a. Enter the following MVS command to display the structures for this data sharing group:

```
D XCF,STRUCTURE,STRNAME=grpname*
```

b. For group buffer pools and the lock structure, enter the following command to force the connections off those structures:

```
SETXCF FORCE,CONNECTION,STRNAME=strname,CONNAME=ALL
```

Connections for the SCA are not held at termination, so there are no SCA connections to force off.

c. Delete all the DB2 coupling facility structures by using the following command for each structure:

```
SETXCF FORCE,STRUCTURE,STRNAME=strname
```

This step is necessary to clean out old information that exists in the coupling facility from your practice startup when you installed the group.

2. If an integrated catalog facility catalog does not already exist, run job DSNTIJCA to create a user catalog.
3. Use the access method services IMPORT command to import the integrated catalog facility catalog.
4. Restore DB2 libraries, such as DB2 reslibs, SMP libraries, user program libraries, user DBRM libraries, CLISTs, SDSNSAMP, or where the installation jobs are, JCL for user-defined table spaces, and so on.
5. Use IDCAMS DELETE NOSCRATCH to delete all catalog and user objects. (Because step 3 imports a user ICF catalog, the catalog reflects data sets that do not exist on disk.)

   Obtain a copy of installation job DSNTIJIN. This job creates DB2 VSAM and non-VSAM data sets. Change the volume serial numbers in the job to volume serial numbers that exist at the recovery site. Comment out the steps that create DB2 non-VSAM data sets, if these data sets already exist. Run DSNTIJIN.

   **Data sharing**

   Obtain a copy of the installation job DSNTIJIN for the first data sharing member to be migrated. Run DSNTIJIN on the first data sharing member. For subsequent members of the data sharing group, run the DSNTIJIN that defines the BSDS and logs.

6. Recover the BSDS:
   a. Use the access method services REPRO command to restore the contents of one BSDS data set (allocated in the previous step). The most recent BSDS image will be found in the last file (archive log with the highest number) on the latest archive log tape.

      **Data sharing**

      The BSDS data sets on each data sharing member need to be restored.

   b. To determine the RBA range for this archive log, use the print log map utility (DSNJU004) to list the current BSDS contents. Find the most recent archive log in the BSDS listing and add 1 to its ENDRBA value. Use this as the STARTRBA. Find the active log in the BSDS listing that starts with this RBA and use its ENDRBA as the ENDRBA.

      **Data Sharing**

      The LRSNs are also required.

   c. Use the change log inventory utility (DSNJU003) to register this latest archive log tape data set in the archive log inventory of the BSDS just restored. This is necessary because the BSDS image on an archive log tape does not reflect the archive log data set residing on that tape.

**Data sharing**

Running DSNJU003 is critical for data sharing groups. Group buffer pool checkpoint information is stored in the BSDS and needs to be included from the most recent archive log.

After these archive logs are registered, use the print log map utility (DSNJU004) with the GROUP option to list the contents of all the BSDSs. You get output that includes the start and end LRSN and RBA values for the latest active log data sets (shown as NOTREUSABLE). If you did not save the values from the DSNJ003I message, you can get those values from here, as shown in Figure 46 and Figure 47.

```
ACTIVE LOG COPY 1 DATA SETS
  START RBA/LRSN/TIME   END RBA/LRSN/TIME      DATE    LTIME  DATA SET INFORMATION
 --------------------  --------------------  --------  -----  --------------------
   000001C20000           000001C67FFF       1996.358  17:25  DSN=DSNDB0G.DB1G.LOGCOPY1.DS03
     ADFA0FB26C6D           ADFA208AA36B                      STATUS=TRUNCATED, REUSABLE
 1996.361  23:37:48.4  1996.362  00:53:10.1
   000001C68000           000001D4FFFF       1996.358  17:25  DSN=DSNDB0G.DB1G.LOGCOPY1.DS01
     ADFA208AA36C           AE3C45273A77                      STATUS=TRUNCATED, NOTREUSABLE
 1996.362  00:53:10.1  1997.048  15:28:23.5
   000001D50000           0000020D3FFF       1996.358  17:25  DSN=DSNDB0G.DB1G.LOGCOPY1.DS02
     AE3C45273A78           ............                      STATUS=NOTREUSABLE
 1997.048  15:28:23.5  ........  ..........
```

*Figure 46. BSDS contents (partial) of member DB1G*

```
ACTIVE LOG COPY 1 DATA SETS
  START RBA/LRSN/TIME   END RBA/LRSN/TIME      DATE    LTIME  DATA SET INFORMATION
 --------------------  --------------------  --------  -----  --------------------
                 EMPTY DATA SET              1996.361  14:14  DSN=DSNDB0G.DB2G.LOGCOPY1.DS03
     000000000000           000000000000                      STATUS=NEW, REUSABLE
 0000.000  00:00:00.0  0000.000  00:00:00.0
   000000000000           0000000D6FFF       1996.361  14:14  DSN=DSNDB0G.DB2G.LOGCOPY1.DS01
     ADFA00BB70FB           AE3C45276DD7                      STATUS=TRUNCATED, NOTREUSABLE
 1996.361  22:30:51.4  1997.048  15:28:23.7
   0000000D7000           00000045AFFF       1996.361  14:14  DSN=DSNDB0G.DB2G.LOGCOPY1.DS02
     AE3C45276DD8           ............                      STATUS=NOTREUSABLE
 1997.048  15:28:23.7  ........  ..........
```

*Figure 47. BSDS contents (partial) of member DB2G*

**Data sharing**

Do all other preparatory activities as you would for a single system. Do these activities for *each* member of the data sharing group.

d. Use the change log inventory utility to adjust the active logs:

   1) Use the DELETE option of the change log inventory utility (DSNJU003) to delete all active logs in the BSDS. Use the BSDS listing produced in the step above to determine the active log data set names.

        2) Use the NEWLOG statement of the change log inventory utility (DSNJU003) to add the active log data sets to the BSDS. Do not specify a STARTRBA or ENDRBA value in the NEWLOG statement. This indicates to DB2 that the new active logs are empty.

    e. If you are using the DB2 distributed data facility, run the change log inventory utility with the DDF statement to update the LOCATION and the LUNAME values in the BSDS.

    f. Use the print log map utility (DSNJU004) to list the new BSDS contents and ensure that the BSDS correctly reflects the active and archive log data set inventories. In particular, ensure that:

        - All active logs show a status of NEW and REUSABLE
        - The archive log inventory is complete and correct (for example, the start and end RBAs should be correct).

    g. If you are using dual BSDSs, make a copy of the newly restored BSDS data set to the second BSDS dataset.

7. Optionally, you can restore archive logs to disk. Archive logs are typically stored on tape, but restoring them to disk could speed later steps. If you elect this option, and the archive log data sets are not cataloged in the primary integrated catalog facility catalog, use the change log inventory utility to update the BSDS. If the archive logs are listed as cataloged in the BSDS, DB2 allocates them using the integrated catalog and not the unit or volser specified in the BSDS. If you are using dual BSDSs, remember to update both copies.

8. Use the DSN1LOGP utility to determine which transactions were in process at the end of the last archive log. Use the following job control language:

```
//SAMP  EXEC PGM=DSN1LOGP
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//ARCHIVE  DD DSN=last-archive,DISP=(OLD,KEEP),UNIT=TAPE,
             LABEL=(2,SL),VOL=SER=volser1
             (NOTE FILE 1 is BSDS COPY)
//SYSIN    DD *
  STARTRBA(yyyyyyyyyyyy) SUMMARY(ONLY)
/*
```

   where *yyyyyyyyyyyy* is the STARTRBA of the last complete checkpoint within the RBA range on the last archive log from the previous print log map.

   DSN1LOGP gives a report. For sample output and information about how to read it, see Part 3 of *DB2 Utility Guide and Reference*.

   Note whether any utilities were executing at the end of the last archive log. You will have to determine the appropriate recovery action to take on each table space involved in a utility job.

   If DSN1LOGP showed that utilities are inflight (PLAN=DSNUTIL), you need SYSUTILX to identify the utility status and determine the recovery approach. See “What to do about utilities in progress” on page 457.

9. Modify DSNZP*xxx* parameters:

    a. Run the DSNTINST CLIST in UPDATE mode. See Part 2 of *DB2 Installation Guide*.

    b. To defer processing of all databases, select “Databases to Start Automatically” from panel DSNTIPB. You are presented with panel DSNTIPS. Type DEFER in the first field, ALL in the second, and press Enter. You are returned to DSNTIPB.

   c. To specify where you are recovering, select “Operator Functions” from panel DSNTIPB. You are presented with panel DSNTIPO. Type RECOVERYSITE in the SITE TYPE field. Press Enter to continue.
   d. To optionally specify which archive log to use, select “Operator Functions” from panel DSNTIPB. You are presented with panel DSNTIPO. Type YES in the READ ARCHIVE COPY2 field if you are using dual archive logging and want to use the second copy of the archive logs. Press Enter to continue.
   e. Reassemble DSNZP*xxx* using job DSNTIJUZ (produced by the CLIST started in the first step).

   At this point you have the log, but the table spaces have not been recovered. With DEFER ALL, DB2 assumes that the table spaces are unavailable, but does the necessary processing to the log. This step also handles the units of recovery in process.

10. Use the change log inventory utility to create a conditional restart control record. In most cases, you can use this form of the CRESTART statement:

```
CRESTART CREATE,ENDRBA=nnnnnnnnn000,FORWARD=YES,
         BACKOUT=YES
```

   where *nnnnnnnnn*000 equals a value one more than the ENDRBA of the latest archive log.

**Data sharing**

If you are recovering a data sharing group, and your logs are not at a single point of consistency, use this form of the CRESTART statement:

```
CRESTART CREATE,ENDLRSN=nnnnnnnnnnnn,FORWARD=YES,BACKOUT=YES
```

where *nnnnnnnnnnnn* is the LRSN of the last log record to be used during restart. Use the same LRSN for **all** members in a data sharing group. Determine the ENDLRSN value using one of the following methods:

- Use the DSN1LOGP summary utility to obtain the ENDLRSN value. In the ’Summary of Completed Events’ section, find the lowest LRSN value listed in the DSN1213I message, for the data sharing group. Use this value for the ENDLRSN in the CRESTART statement.
- Use the print log map utility (DSNJU004) to list the BSDS contents. Find the ENDLRSN of the last log record available for each active member of the data sharing group. Subtract 1 from the **lowest** ENDLRSN in the data sharing group. Use this value for the ENDLRSN in the CRESTART statement. (In our example in Figure 46 on page 451, that is AE3C45273A77 - 1, which is AE3C45273A76.)
- If only the console logs are available, use the archive offload message, DSNJ003I to obtain the ENDLRSN. Compare the ending LRSN values for all members’ archive logs. Subtract 1 from the **lowest** LRSN in the data sharing group. Use this value for the ENDLRSN in the CRESTART statement. (In our example in Figure 46 on page 451, that is AE3C45273A77 - 1, which is AE3C45273A76.)

DB2 discards any log information in the bootstrap data set and the active logs with an RBA greater than or equal to *nnnnnnnnn*000 or an LRSN greater than *nnnnnnnnnnnn* as listed in the CRESTART statements above.

Use the print log map utility to verify that the conditional restart control record that you created in the previous step is active.

11. Enter the command START DB2 ACCESS(MAINT).

**Data Sharing**

If there is a discrepancy among the print log map reports as to the number of members in the group, record the one that shows the highest number of members. (This is an unlikely occurrence.) Start this DB2 first using ACCESS(MAINT). DB2 will prompt you to start each additional DB2 subsystem in the group.

After all additional members are successfully restarted, and if you are going to run single-system data sharing at the recovery site, stop all DB2s but one by using the STOP DB2 command with MODE(QUIESCE).

If you planned to use the light mode when starting the DB2 group, add the LIGHT parameter to the START command listed above. Start the members that run in LIGHT(NO) mode first, followed by the LIGHT(YES) members. See “Preparing for disaster recovery” on page 385 for details on using restart light at a recovery site.

Even though DB2 marks all table spaces for deferred restart, log records are written so that in-abort and inflight units of recovery are backed out. In-commit units of recovery are completed, but no additional log records are written at restart to cause this. This happens when the original redo log records are applied by the RECOVER utility.

At the primary site, DB2 probably committed or aborted the inflight units of recovery, but you have no way of knowing.

During restart, DB2 accesses two table spaces that result in DSNT501I, DSNT500I, and DSNL700I resource unavailable messages, regardless of DEFER status. The messages are normal and expected, and you can ignore them.

The return code accompanying the message might be one of the following, although other codes are possible:

**00C90081**
This return code occurs if there is activity against the object during restart as a result of a unit of recovery or pending writes. In this case the status shown as a result of -DISPLAY is STOP,DEFER.

**00C90094**
Because the table space is currently only a defined VSAM data set, it is in an unexpected state to DB2.

**00C900A9**
This codes indicates that an attempt was made to allocate a deferred resource.

12. Resolve the indoubt units of recovery.

The RECOVER utility, which you will soon invoke, will fail on any table space that has indoubt units of recovery. Because of this, you must resolve them first. Determine the proper action to take (commit or abort) for each unit of recovery. To resolve indoubt units of recovery, see “Resolving indoubt units of recovery” on page 363 on page 363. From an install SYSADM authorization ID, enter the RECOVER INDOUBT command for all affected transactions.

13. To recover the catalog and directory, follow these instructions:

    The RECOVER function includes: RECOVER TABLESPACE, RECOVER INDEX, or REBUILD INDEX. If you have an image copy of an index, use RECOVER INDEX. If you do not have an image copy of an index, use REBUILD INDEX to reconstruct the index from the recovered table space.

    a. Recover DSNDB01.SYSUTILX. This must be a separate job step.
    b. Recover all indexes on SYSUTILX. This must be a separate job step.
    c. Your recovery strategy for an object depends on whether a utility was running against it at the time the latest archive log was created. To identify the utilities that were running, you must recover SYSUTILX.

       You cannot restart a utility at the recovery site that was interrupted at the disaster site. You must use the TERM command to terminate it. The TERM UTILITY command can be used on any object except DSNDB01.SYSUTILX.

       Determine which utilities were executing and the table spaces involved by following these steps:

       1) Enter the DISPLAY UTILITY(*) command and record the utility and the current phase.
       2) Run the DIAGNOSE utility with the DISPLAY SYSUTILX statement. The output consists of information about each active utility, including the table space name (in most instances). It is the only way to correlate the object name with the utility. Message DSNU866I gives information on the utility, while DSNU867I gives the database and table space name in USUDBNAM and USUSPNAM respectively.
    d. Use the command TERM UTILITY to terminate any utilities in progress on catalog or directory table spaces.

       See “What to do about utilities in progress” on page 457 for information on how to recover catalog and directory table spaces on which utilities were running.
    e. Recover the rest of the catalog and directory objects starting with DBD01, in the order shown in the description of the RECOVER utility in Part 2 of *DB2 Utility Guide and Reference*.
14. Use any method desired to verify the integrity of the DB2 catalog and directory. Migration step 1 in Chapter 1 of *DB2 Installation Guide* lists one option for verification. The catalog queries in member DSNTESQ of data set DSN710.SDSNSAMP can be used after the work file database is defined and initialized.
15. Define and initialize the work file database.
    a. Define temporary work files. Use installation job DSNTIJTM as a model.
    b. Issue the command -START DATABASE(*work-file-database*) to start the work file database.
16. If you use data definition control support, recover the objects in the data definition control support database.
17. If you use the resource limit facility, recover the objects in the resource limit control facility database.
18. Modify DSNZP*xxx* to restart all databases:
    a. Run the DSNTINST CLIST in UPDATE mode. See Part 2 of *DB2 Installation Guide* .

   b. From panel DSNTIPB select “Databases to Start Automatically”. You are presented with panel DSNTIPS. Type RESTART in the first field, ALL in the second and press Enter. You are returned to DSNTIPB.

   c. Reassemble DSNZP*xxx* using job DSNTIJUZ (produced by the CLIST started in the first step).

19. Stop and start DB2.

20. Make a full image copy of the catalog and directory.

21. Recover user table spaces and index spaces. See “What to do about utilities in progress” on page 457 for information on how to recover table spaces or index spaces on which utilities were running. You cannot restart a utility at the recovery site that was interrupted at the disaster site. Use the TERM command to terminate any utilities running against user table spaces or index spaces.

    a. To determine which, if any, of your table spaces or index spaces are user-managed, perform the following queries for table spaces and index spaces.

       Table spaces:

       ```
       SELECT * FROM SYSIBM.SYSTABLEPART WHERE STORTYPE='E';
       ```

       Index spaces:

       ```
       SELECT * FROM SYSIBM.SYSINDEXPART WHERE STORTYPE='E';
       ```

       To allocate user-managed table spaces or index spaces, use the access method services DEFINE CLUSTER command.To find the correct IPREFIX for the DEFINE CLUSTER command, perform the following queries for table spaces and index spaces.

       Table spaces:

       ```
       SELECT DBNAME, TSNAME, PARTITION, IPREFIX FROM SYSIBM.SYSTABLEPART
       WHERE DBNAME=dbname AND TSNAME=tsname
       ORDER BY PARTITION;
       ```

       Index spaces:

       ```
       SELECT IXNAME, PARTITION, IPREFIX FROM SYSIBM.SYSINDEXPART
       WHERE IXCREATOR=ixcreator AND IXNAME=ixname
       ORDER BY PARTITION;
       ```

       Now you can perform the DEFINE CLUSTER command with the correct IPREFIX (I or J) in the data set name:

       ```
       catname.DSNDBC.dbname.spname.y0001.A00x
       ```

       where *y* can be either I or J, *x* is C (for VSAM clusters) or D (for VSAM data components), and *spname* is either the table space or index space name.

       Access method services commands are described in detail in *DFSMS/MVS: Access Method Services for VSAM Catalogs*.

    b. If your user table spaces or index spaces are STOGROUP-defined, and if the volume serial numbers at the recovery site are different from those at the local site, use ALTER STOGROUP to change them in the DB2 catalog.

    c. Recover all user table spaces and index spaces from the appropriate image copies. If you do not copy your indexes, use the REBUILD INDEX utility to reconstruct the indexes.

    d. Start all user table spaces and index spaces for read or write processing by issuing the command -START DATABASE with the ACCESS(RW) option.
    e. Resolve any remaining check pending states that would prevent COPY execution.
    f. Run select queries with known results.

22. Make full image copies of all table spaces and indexes with the COPY YES attribute.
23. Finally, compensate for lost work since the last archive was created by rerunning online transactions and batch jobs.

***What to do about utilities in progress:*** If any utility jobs were running after the last time that the log was offloaded before the disaster, you might need to take some additional steps. After restarting DB2, the following utilities only need to be terminated with the TERM UTILITY command:
- CHECK INDEX
- MERGECOPY
- MODIFY
- QUIESCE
- RECOVER
- RUNSTATS
- STOSPACE

It is preferable to allow the RECOVER utility to reset pending states. However, it is occasionally necessary to use the REPAIR utility to reset them. Do not start the table space with ACCESS(FORCE) because FORCE resets any page set exception conditions described in “Database page set control records” on page 962.

For the following utility jobs, perform the actions indicated:

**CHECK DATA**
Terminate the utility and run it again after recovery is complete.

**COPY** After you enter the TERM command, DB2 places a record in the SYSCOPY catalog table indicating that the COPY utility was terminated. This makes it necessary for you to make a full image copy. When you copy your environment at the completion of the disaster recovery scenario, you fulfill that requirement.

**LOAD** Find the options you specified in Table 69, and perform the specified actions.

*Table 69. Actions when LOAD is interrupted*

| LOAD options specified | What to do |
|---|---|
| LOG YES | If the RELOAD phase completed, then recover to the current time, and recover the indexes.<br>If the RELOAD phase did not complete, then recover to a prior point in time. The SYSCOPY record inserted at the beginning of the RELOAD phase contains the RBA or LRSN. |
| LOG NO<br>*copy spec* | If the RELOAD phase completed, then the table space is complete after you recover it to the current time. Recover the indexes.<br>If the RELOAD phase did not complete, then recover the table space to a prior point in time. Recover the indexes. |

*Table 69. Actions when LOAD is interrupted (continued)*

| LOAD options specified | What to do |
|---|---|
| LOG NO<br>*copy spec*<br>SORTKEYS | If the BUILD or SORTBLD phase completed, then recover to the current time, and recover the indexes.<br>If the BUILD or SORTBLD phase did not complete, then recover to a prior point in time. Recover the indexes. |
| LOG NO | Recover the table space to a prior point in time. You can use TOCOPY to do this. |

To avoid extra loss of data in a future disaster situation, run QUIESCE on table spaces before invoking LOAD. This enables you to recover a table space using TORBA instead of TOCOPY.

**REORG**

For a **user** table space, find the options you specified in Table 70, and perform the specified actions.

*Table 70. Actions when REORG is interrupted*

| REORG options specified | What to do |
|---|---|
| LOG YES | If the RELOAD phase completed, then recover to the current time, and recover the indexes.<br>If the RELOAD phase did not complete, then recover to the current time to restore the table space to the point before REORG began. Recover the indexes. |
| LOG NO | If the RELOAD phase completed, then recover to a prior point in time. You can use TOCOPY or TORBA to do this.<br>If the RELOAD phase did not complete, then recover to the current time to restore the table space to the point before REORG began. Recover the indexes. |
| LOG NO<br>*copy spec* | If the RELOAD phase completed, then the table space is complete after you recover it to the current time. Recover the indexes.<br>If the RELOAD phase did not complete, then recover to the current time to restore table space to the point before REORG began. Recover the indexes. |
| LOG NO<br>*copy spec*<br>SORTKEYS | If the build or SORTBLD phase completed, then recover to the current time, and recover the indexes.<br>If the build or SORTBLD phase did not complete, then recover to the current time to restore the table space to the point before REORG began. Recover the indexes. |
| SHRLEVEL CHANGE | If the SWITCH phase completed, terminate the utility. Recover the table space to the current time. Recover the indexes.<br>If the SWITCH phase did not complete, recover the table space to the current time. Recover the indexes. |
| SHRLEVEL REFERENCE | Same as for SHRLEVEL CHANGE. |

For a **catalog or directory** table space, follow these instructions:

Table spaces with links cannot use online REORG. For those table spaces that can use online REORG, find the options you specified in Table 70 on page 458, and perform the specified actions.

If you have no image copies from immediately before REORG failed, use this procedure:

1. From your DISPLAY UTILITY and DIAGNOSE output, determine what phase REORG was in and which table space it was reorganizing when the disaster occurred.
2. Run RECOVER on the catalog and directory in the order shown in Part 2 of *DB2 Utility Guide and Reference*. Recover all table spaces to the current time, *except* the table space that was being reorganized. If the RELOAD phase of the REORG on that table space had not completed when the disaster occurred, recover the table space to the current time. Because REORG does not generate any log records prior to the RELOAD phase for catalog and directory objects, the RECOVER to current restores the data to the state it was in before the REORG. If the RELOAD phase completed, do the following:
   a. Run DSN1LOGP against the archive log data sets from the disaster site.
   b. Find the begin-UR log record for the REORG that failed in the DSN1LOGP output.
   c. Run RECOVER with the TORBA option on the table space that was being reorganized. Use the URID of the begin-UR record as the TORBA value.
3. Recover or rebuild all indexes.

If you have image copies from immediately before REORG failed, run RECOVER with the option TOCOPY to recover the catalog and directory, in the order shown in Part 2 of *DB2 Utility Guide and Reference*.

***Recommendation:*** Make full image copies of the catalog and directory before you run REORG on them.

## Using a tracker site for disaster recovery

This section describes a different method for disaster recovery from that described in “Remote site recovery from disaster at a local site” on page 449. Many steps are similar to a regular disaster recovery, so these steps are not described in detail here.

***Recommendation:*** Test and document a disaster procedure that is customized for your location.

***Overview of the method:*** A DB2 *tracker* site is a separate DB2 subsystem or data sharing group that exists solely for the purpose of keeping shadow copies of your primary site’s data. No independent work can be run on the tracker site.

From the primary site, you transfer the BSDS and the archive logs, and that tracker site runs periodic LOGONLY recoveries to keep the shadow data up-to-date. If a disaster occurs at the primary site, the tracker site becomes the *takeover* site. Because the tracker site has been shadowing the activity on the primary site, you do not have to constantly ship image copies; the takeover time for the tracker site might be faster because DB2 recovery does not have to use image copies.

The following topics are described in this section:

"Characteristics of a tracker site"
"Setting up a tracker site"
"Establishing a recovery cycle at the tracker site" on page 461
"Maintaining the tracker site" on page 464
"The disaster happens: making the tracker site the takeover site" on page 464

## Characteristics of a tracker site

Because the tracker site must use only the primary site's logs for recovery, you must not update the catalog and directory or the data at the tracker site. The DB2 subsystem at the tracker site disallows updates. In summary:

- The following SQL statements are not allowed at a tracker site:
  - GRANT or REVOKE
  - DROP, ALTER, or CREATE
  - UPDATE, INSERT, or DELETE

  Dynamic read-only SELECT statements are allowed.
- The only online utilities that are allowed are REPORT, DIAGNOSE, RECOVER, and REBUILD. Recovery to a prior point in time is not allowed.
- BIND is not allowed.
- TERM UTIL is not allowed for LOAD, REORG, REPAIR, and COPY.
- The START DATABASE command is not allowed when LPL or GRECP status exists for the object of the command. It is not necessary to use START DATABASE to clear LPL or GRECP conditions, because you are going to be running RECOVERY jobs that clear the conditions.
- The START DATABASE command with ACCESS(FORCE) is not allowed.
- Down-level detection is disabled.
- Log archiving is disabled.

## Setting up a tracker site

To set up the tracker site:

1. Create a mirror image of your primary DB2 subsystem or data sharing group. This process is described in steps 1 through 4 of the normal disaster recovery procedure, and includes such things as creating catalogs and restoring DB2 libraries.
2. Modify the subsystem parameters as follows:
   - Set the TRKSITE subsystem parameter to YES.
   - Optionally, set the SITETYP parameter to RECOVERYSITE if the full image copies this site will be receiving are created as remote site copies.
3. Use the access method services DEFINE CLUSTER command to allocate data sets for all user-managed table spaces that you will be sending over from the primary site. Similarly, allocate data sets for any user-managed indexes that you want to rebuild during recovery cycles. The main reason to rebuild indexes for recovery cycles is for running queries on the tracker site. If you do not require indexes, you do not have to rebuild them for recovery cycles. For nonpartitioning indexes on very large tables, you can include indexes for LOGONLY recovery during the recovery cycle, which can reduce the amount of time it takes to bring up the disaster site. Be sure you define data sets with the proper **I** or **J** prefix for both indexes and table spaces.
4. Send full image copies of all the primary site's DB2 data to the tracker site.
5. Tailor installation job DSNTIJIN to create DB2 catalog data sets.

***Important:*** Do not attempt to start the tracker site when you are setting it up. You must follow the procedure described in “Establishing a recovery cycle at the tracker site”.

## Establishing a recovery cycle at the tracker site

When the tracker site has full image copies of all the data at the primary site, you periodically send the archive logs and BSDSs from the primary site to the tracker site and recover data from the log.

The cycle of events is:

1. While your primary site continues its usual workload, send a copy of the primary site’s BSDSs and archive logs to the tracker site.

   Send full image copies for any object when:

   - The table space or partition is reorganized, loaded, or repaired with the LOG NO option
   - The object has undergone a point-in-time recovery

   See “What to do about DSNDB01.SYSUTILX” on page 463 for information about options for preparing SYSUTILX for recovery.

   ***Recommendation:*** If you are taking incremental image copies, run the MERGECOPY utility at the primary site before sending the copy to the tracker site.
2. At the tracker site, restore the BSDS that was received from the primary site. Find the most recent BSDS image on the latest archive log tape and use the change log inventory utility (DSNJU003) to register the latest archive log tape in the archive log inventory of this BSDS. You must also delete the tracker site’s active logs and add new empty active logs to the BSDS inventory.

   For more details on this step, see step 6 of “Remote site recovery from disaster at a local site” on page 449.
3. Use the change log inventory utility (DSNJU003) with a CRESTART statement that looks like this:

   ```
   CRESTART CREATE,ENDRBA=nnnnnnnnn000,FORWARD=NO,BACKOUT=NO
   ```

   where *nnnnnnnnn*000 equals ENDRBA + 1 of the latest archive log. You must not specify STARTRBA, because you cannot cold start or skip logs in a tracker system.

   **Data sharing**

   If you are recovering a data sharing group, you **must** use this form of the CRESTART statement on all members of the data sharing group. The ENDLRSN value must be the same for all members:

   ```
   CRESTART CREATE,ENDRBA=nnnnnnnnnnnn,FORWARD=NO,BACKOUT=NO
   ```

   where *nnnnnnnnnnnn* is the lowest ENDLRSN of all the members to be read during restart. The ENDLRSN value must be the same:

   - If you get the ENDLRSN from the output of the print log map utility (DSNJU004) or from the console logs using message DSNJ003I, you must use ENDLRSN-1 as the input to the conditional restart.
   - If you get the ENDLRSN from the output of the DSN1LOGP utility (DSN1213I message), you can use the value as is.

The ENDLRSN or ENDRBA value indicates the end log point for data recovery and for truncating the archive log. With ENDLRSN, the *missing* log records between the lowest and highest ENDLRSN values for all the members are applied during the next recovery cycle.

4. If the tracker site is a data sharing group, delete all DB2 coupling facility structures before restarting the tracker members.
5. If you are using LOGONLY recovery for DSNDB01.SYSUTILX, use DSN1COPY to restore SYSUTILX from the previous tracker cycle (or the initial copy if this is the first tracker cycle.)
6. At the tracker site, restart DB2 to begin a **tracker site recovery cycle**.

**Data sharing**

For data sharing, restart **every** member of the data sharing group.

7. At the tracker site, run RECOVER jobs to recover the data from the image copies, if needed, or use the LOGONLY option to recover from the logs received from the primary site to keep the shadow DB2 data current. See “Media failures during LOGONLY recovery” on page 463 for information about what to do if a media failure occurs during LOGONLY recovery.
   a. Recover the catalog and directory

      See *DB2 Utility Guide and Reference* for information about the order of recovery for the catalog and directory objects.

      ***Recovering SYSUTILX:*** If you are doing a LOGONLY recovery on SYSUTILX from a previous DSN1COPY backup, make another DSN1COPY copy of that table space after the LOGONLY recovery is complete and before recovering any other catalog or directory objects.

      After you recover SYSUTILX and either recover or rebuild its indexes, and before recovering other system and user table spaces, find out what utilities were running at the primary site.

      1) Enter DISPLAY UTIL(*) for a list of currently running utilities.
      2) Run the DIAGNOSE utility with the DISPLAY SYSUTIL statement to find out the names of the object on which the utilities are running. Installation SYSOPR authority is required.

      Because the tracker DB2 prevents the TERM UTIL command from removing the status of utilities, the following restrictions apply:

      - If a LOAD, REORG, REPAIR, or COPY is in progress on any catalog or directory object at the primary site, you cannot continue recovering through the list of catalog and directory objects. Therefore, you cannot recover any user data. Shut down and wait until the next recovery cycle when you have a full image copy with which to do recovery.
      - If a LOAD, REORG, REPAIR, or COPY utility is in progress on any user data, you cannot recover that object until the next cycle when you have a full image copy.
      - If an object is in the restart pending state, you can use LOGONLY recovery to recover the object when that object is no longer in restart pending state.

**Data sharing**

If read/write shared data (GPB-dependent data) is in the advisory recovery pending state, the tracker DB2 performs recovery processing. Because the tracker DB2 always performs a conditional restart, the postponed indoubt units of recovery are not recognized after the tracker DB2 restarts.

***User-defined catalog indexes:*** Unless you require them for catalog query performance, it is not necessary to rebuild user-defined catalog indexes until the tracker DB2 becomes the takeover DB2. However, if you are recovering user-defined catalog indexes, do the recover in this step.

b. If needed, recover other system data such as the data definition control support table spaces and the resource limit facility table spaces.

c. Recover user data and, optionally, rebuild your indexes.

   It is not necessary to rebuild indexes unless you intend to run dynamic queries on the data at the tracker site.

Because this is a tracker site, DB2 stores the conditional restart ENDRBA or ENDLRSN in the page set after each recovery completes successfully. By storing the log truncation value in the page set, DB2 ensures that it does not skip any log records between recovery cycles.

8. After all recovery has completed at the tracker site, shut down the tracker site DB2. This is the end of the tracker site recovery cycle.

   If you choose to, you can stop and start the tracker DB2 several times before completing a recovery cycle.

## What to do about DSNDB01.SYSUTILX

DB2 does not write SYSLGRNX entries for DSNDB01.SYSUTILX, which can lead to long recovery times at the tracker site. In addition, SYSUTILX and its indexes are updated during the tracker cycle when you run your recoveries. Because SYSUTILX must remain consistent with the SYSUTILX at the primary site, the tracker cycle updates must be discarded before the next tracker cycle.

There are two ways to plan for recovering SYSUTILX:

- Send a full image copy to DSNDB01.SYSUTILX for each recovery cycle. At the tracker site, you would use normal recovery (that is, full image copy and logs).
- Use DSN1COPY copies of SYSUTILX at the tracker site and LOGONLY recoveries. The sequence of steps is as follows:
  1. Use DSN1COPY to restore a copy made during the last tracker cycle.
  2. Run RECOVER with the LOGONLY option on the table space.
  3. *Before running any other utilities,* use DSN1COPY to make a copy to be used during the next tracker cycle.

## Media failures during LOGONLY recovery

As documented in *DB2 Utility Guide and Reference*, if there is an I/O error during a LOGONLY recovery, recover the object using the image copies and logs after you correct the media failure.

If an entire volume is corrupted and you are using DB2 storage groups, you cannot use the ALTER STOGROUP statement to remove the corrupted volume and add another as is documented for a non-tracker system. Instead, you must remove the corrupted volume and reinitialize another volume with the same volume serial before invoking the RECOVER utility for all table spaces and indexes on that volume.

## Maintaining the tracker site

It is recommended that the tracker site and primary site be at the same maintenance level to avoid unexpected problems. Between recovery cycles, you can apply maintenance as you normally do, by stopping and restarting the DB2 subsystem or DB2 member.

If a tracker site fails, you can restart it normally.

Because bringing up your tracker site as the takeover site destroys the tracker site environment, you should save your complete tracker site prior to takeover site testing. The tracker site can then be restored after the takeover site testing, and the tracker site recovery cycles can be resumed.

**Data sharing group restarts**

During recovery cycles, the first member that comes up puts the ENDLRSN value in the shared communications area (SCA) of the coupling facility. If an SCA failure occurs during a recovery cycle, you must go through the recovery cycle again, using the same ENDLRSN value for your conditional restart.

## The disaster happens: making the tracker site the takeover site

If a disaster occurs at the primary site, the tracker site must become the *takeover* site. After the takeover site is restarted, run RECOVER jobs for log data or image copies that were enroute when the disaster occurred.

1. Restore the BSDS and register the archive log from the last archive you received from the primary site.
2. For scenarios other than data sharing, continue with the next step.

   **Data sharing**

   If this is a data sharing system, delete the coupling facility structures.

3. Ensure that the DEFER ALL and TRKSITE NO subsystem parameters are specified.
4. If this is a non-data-sharing DB2, the log truncation point varies depending on whether you have received more logs from the primary site since the last recovery cycle:
   - If you received no more logs from the primary site:

     Start DB2 using the same ENDRBA you used on the last tracker cycle. Specify FORWARD=YES and BACKOUT=YES (this takes care of uncommitted work). If you have fully recovered the objects during the previous cycle, then they are current except for any objects that had outstanding units of recovery during restart. Because the previous cycle specified NO for FORWARD and BACKOUT and you have now specified YES, affected data sets are placed in LPL. Restart the objects that are in LPL status using the following START DATABASE command:

     ```
     START DATABASE(*) SPACENAM(*)
     ```

After you issue the command, all table spaces and indexes that were previously recovered are now current. Remember to rebuild any indexes that were not recovered during the previous tracker cycle, including user-defined indexes on the DB2 catalog.

- If you received more logs from the primary site:

  Start DB2 using the truncated RBA *nnnnnnnnn*000, which is the ENDRBA + 1 of the latest archive log. Specify FORWARD=YES and BACKOUT=YES. Run your recoveries as you did during recovery cycles.

**Data sharing**

You must restart **every** member of the data sharing group, using this form of the CRESTART statement:

```
CRESTART CREATE,ENDLRSN=nnnnnnnnnnnn,FORWARD=YES,BACKOUT=YES
```

where *nnnnnnnnnnnn* is the LRSN of the last log record to be used during restart. See step 3 of “Establishing a recovery cycle at the tracker site” on page 461 for more information about determining this value. The takeover DB2s must specify conditional restart with a common ENDLRSN value to allow all remote members to logically truncate the logs at a consistent point.

5. As described for a tracker recovery cycle, recover SYSUTILX from an image copy from the primary site, or from a previous DSN1COPY taken at the tracker site.
6. Terminate any in-progress utilities using the following steps:
   a. Enter the command DISPLAY UTIL(*).
   b. Run the DIAGNOSE utility with DISPLAY SYSUTIL to get the names of objects on which utilities are being run.
   c. Terminate in-progress utilities using the command TERM UTIL(*).

      See “What to do about utilities in progress” on page 457 for more information about how to terminate in-progress utilities and how to recover an object on which a utility was running.
7. Continue with your recoveries either with the LOGONLY option or image copies. Do not forget to rebuild indexes, including IBM and user-defined indexes on the DB2 catalog and user-defined indexes on table spaces.

## Resolving indoubt threads

This section describes problem scenarios involving indoubt threads resulting from the following types of error conditions:

“Communication failure between two systems” on page 467
“Making a heuristic decision” on page 468
“IMS outage that results in an IMS cold start” on page 469
“DB2 outage at a requester results in a DB2 cold start” on page 469
“DB2 outage at a server results in a DB2 cold start” on page 472
“Correcting a heuristic decision” on page 473

All scenarios are developed in the context presented in “Description of the environment” on page 466. System programmer, operator, and database administrator actions are indicated for the examples as appropriate. In these descriptions, the term *administrator* refers to the database administrator (DBA) if not otherwise specified.

## Description of the environment

### Configuration

The configuration is composed of four systems at three geographic locations: Seattle (SEA), San Jose (SJ) and Los Angeles (LA). The system descriptions are as follows:

- DB2 subsystem at Seattle, Location name = IBMSEADB20001, Network name = IBM.SEADB21
- DB2 subsystem at San Jose, Location name = IBMSJ0DB20001 Network name = IBM.SJDB21
- DB2 subsystem at Los Angeles, Location name = IBMLA0DB20001 Network name = IBM.LADB21
- IMS subsystem at Seattle, Connection name = SEAIMS01

### Applications

The following IMS and TSO applications are running at Seattle and accessing both local and remote data.

- IMS application, IMSAPP01, at Seattle, accessing local data and remote data by DRDA access at San Jose, which is accessing remote data on behalf of Seattle by DB2 private protocol access at Los Angeles.
- TSO application, TSOAPP01, at Seattle, accessing data by DRDA access at San Jose and at Los Angeles.

### Threads

The following threads are described and keyed to Figure 48 on page 467. Data base access threads (DBAT) access data on behalf of a thread (either allied or DBAT) at a remote requester.

- Allied IMS thread **A** at Seattle accessing data at San Jose by DRDA access.
  - DBAT at San Jose accessing data for Seattle by DRDA access **1** and requesting data at Los Angeles by DB2 private protocol access **2**.
  - DBAT at Los Angeles accessing data for San Jose by DB2 private protocol access **2**.
- Allied TSO thread **B** at Seattle accessing local data and remote data at San Jose and Los Angeles, by DRDA access.
  - DBAT at San Jose accessing data for Seattle by DRDA access **3**.
  - DBAT at Los Angeles accessing data for Seattle by DRDA access **4**.

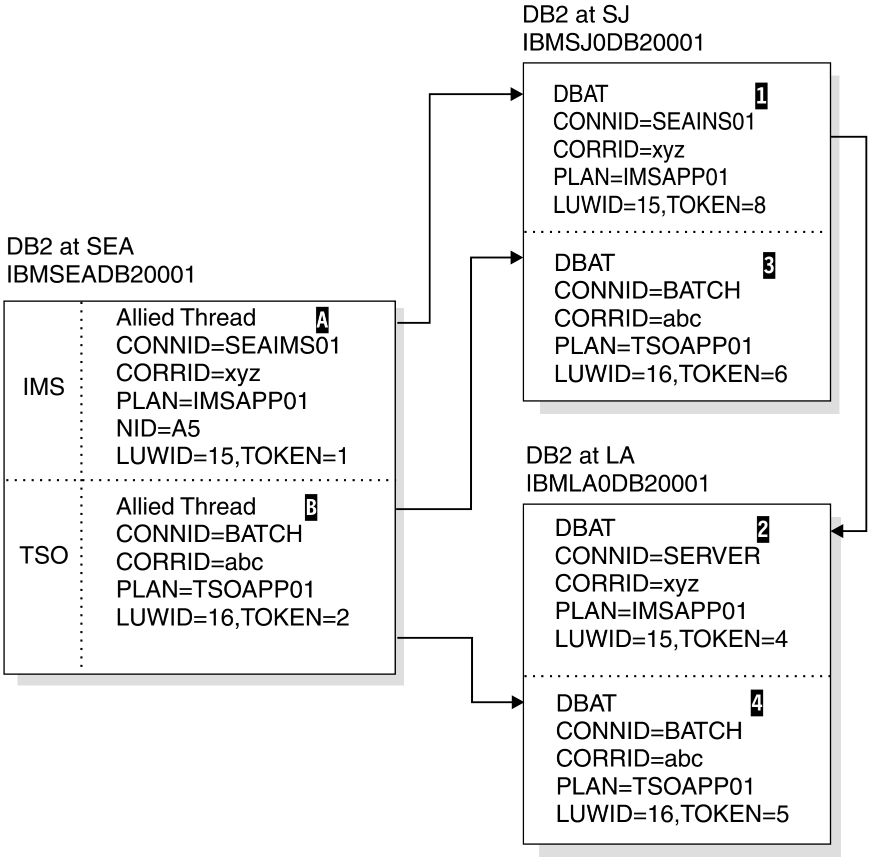


*Figure 48. Resolving indoubt threads. Results of issuing -DIS THD TYPE(ACTIVE) at each DB2 system.*

The results of issuing the DISPLAY THREAD TYPE(ACTIVE) command to display the status of threads at all DB2 locations are summarized in the boxes of Figure 48. The logical unit of work IDs (LUWIDs) have been shortened for readability:

- LUWID=15 would be IBM.SEADB21.15A86A876789.0010
- LUWID=16 would be IBM.SEADB21.16B57B954427.0003

For the purposes of this section, assume that both applications have updated data at all DB2 locations. In the following problem scenarios, the error occurs after the coordinator has recorded the commit decision, but before the affected participants have recorded the commit decision. These participants are therefore indoubt.

## Communication failure between two systems

***Problem:*** A communication failure occurred between Seattle and Los Angeles after the DBAT at LA completed phase 1 of commit processing. At SEA, the TSO thread, LUWID=16 and TOKEN=2 **B**, cannot complete the commit with the DBAT at LA **4**.

***Symptom:*** At SEA, NetView alert A006 is generated and message DSNL406 is displayed, indicating an indoubt thread at LA because of communication failure. At LA, alert A006 is generated and message DSNL405 is displayed, indicating a thread has entered indoubt state because of communication failure with SEA.

***System action:*** At SEA, an IFCID 209 trace record is written. After the alert has been generated, and after the message has been displayed, the thread completes the commit, which includes the DBAT at SJ 3 . Concurrently, the thread is added to the list of threads for which the SEA DB2 has an indoubt resolution responsibility. The thread appears in a display thread report for indoubt threads. The thread also appears in a display thread report for active threads until the application terminates.

The TSO application is told that the commit succeeded. If the application continues and processes another SQL request, it is rejected with an SQL code indicating it must roll back before any more SQL requests can be processed. This is to insure that the application does not proceed with an assumption based upon data retrieved from LA, or with the expectation that cursor positioning at LA is still intact.

At LA, an IFCID 209 trace record is written. After the alert is generated and the message displayed, the DBAT 4 is placed into the indoubt state. All locks remain held until resolution occurs. The thread appears in a display thread report for indoubt threads.

The DB2 systems, at both SEA and LA, periodically attempt reconnecting and automatically resolving the indoubt thread. If the communication failure only affects the session being used by the TSO application, and other sessions are available, automatic resolution occurs in a relatively short time. At this time, message DSNL407 is displayed by both DB2 subsystems.

***Operator action:*** If message DSNL407 or DSNL415 for the thread identified in message DSNL405 does not appear in a reasonable period of time, call the system programmer. A communication failure is making database resources unavailable.

***System programmer action:*** Determine and correct the cause of the communication failure. When corrected, automatic resolution of the indoubt thread occurs within a short time. If the failure cannot be corrected for a long time, call the database administrator. The database administrator might want to make a heuristic decision to release the database resources held for the indoubt thread. See “Making a heuristic decision”.

## Making a heuristic decision

***Problem:*** The indoubt thread at LA is holding database resources that are needed by other applications.

***Symptom:*** Many symptoms can be present, including:

- Message DSNL405 indicating a thread in the indoubt state.
- A display thread report of active threads showing a larger than normal number of threads.
- A display thread report of indoubt threads continuing to show the same thread.
- A display database report with the LOCKS option showing a large number of threads waiting for the locks held by the indoubt thread.
- Some threads terminating due to time out.
- IMS and CICS transactions not completing.

***Database administrator action:*** Determine whether to commit or abort the indoubt thread. First, determine the name of the commit coordinator for the indoubt thread. This is the location name of the DB2 subsystem at SEA, and is included in the DB2 indoubt thread display report at LA. Then, have an authorized person at SEA perform one of the following:

- If the coordinator DB2 subsystem is active, or can be started, request a display thread report for indoubt threads, specifying the LUWID of the thread.

(Remember that the token used at LA is different than the token used at SEA). *If there is no report entry for the LUWID, then the proper action is to abort. If there is an entry for the LUWID, it shows the proper action to take.*

- If the coordinator DB2 subsystem is not active and cannot be started, and if statistics class 4 was active when DB2 was active, search the SEA SMF data for an IFCID 209 event entry containing the indoubt LUWID. This entry indicates whether the commit decision was commit or abort.
- If statistics class 4 is not available, then run, at SEA, the DSN1LOGP utility requesting a summary report. The volume of log data to be searched can be restricted if you can determine the approximate SEA log RBA value in effect at the time of the communication failure. A DSN1LOGP entry in the summary report for the indoubt LUWID indicates whether the decision was commit or abort.

After determining the correct action to take, issue the -RECOVER INDOUBT command at the LA DB2 subsystem, specifying the LUWID and the correct action.

***System action:*** Issuing the RECOVER INDOUBT command at LA results in committing or aborting the indoubt thread. Locks are released. The thread does not disappear from the indoubt thread display until resolution with SEA is completed. The recover indoubt report shows that the thread is either committed or aborted by a heuristic decision. An IFCID 203 trace record is written, recording the heuristic action.

## IMS outage that results in an IMS cold start

***Problem:*** The abnormal termination of IMS has left one allied thread **A** at the SEA DB2 subsystem indoubt. This is the thread having LUWID=15. Because the SEA DB2 still has effective communication with the DB2 subsystem at SJ, the LUWID=15 DBAT **1** at this system is waiting for the SEA DB2 to communicate the final decision and is not aware that IMS has failed. Also, the LUWID=15 DBAT at LA **2** which is connected to SJ is also waiting for SJ to communicate the final decision. This cannot be done until SEA communicates the decision to SJ.

***Symptom:*** When IMS is cold started, and later reconnects with the SEA DB2 subsystem, IMS is not able to resolve the indoubt thread with DB2. Message DSNM004I is displayed at the IMS master terminal. This is the same process as described in “Resolution of indoubt units of recovery” on page 414.

***System action:*** This is the same process as described in “Resolution of indoubt units of recovery” on page 414.

***System programmer action:*** This is the same process as described in “Resolution of indoubt units of recovery” on page 414.

When the indoubt thread at the SEA DB2 subsystem is resolved by issuing the RECOVER INDOUBT command, completion of the two-phase commit process with the DB2 subsystems at SJ and LA occurs, and the unit of work commits or aborts.

***Operator action:*** This is the same process as described in “Resolution of indoubt units of recovery” on page 414.

## DB2 outage at a requester results in a DB2 cold start

***Problem:*** The abnormal termination of the SEA DB2 has left the two DBATs at SJ **1**, **3** and the LUWID=16 DBAT at LA **4** indoubt. The LUWID=15 DBAT at LA **2**, connected to SJ, is waiting for the SJ DB2 to communicate the final decision.

The IMS subsystem at SEA is operational and has the responsibility of resolving indoubt units with the SEA DB2.

***Symptom:*** The DB2 subsystem at SEA is started with a conditional restart record in the BSDS indicating a cold start:

- When the IMS subsystem reconnects, it attempts to resolve the indoubt thread identified in IMS as NID=A5. IMS has a resource recovery element (RRE) for this thread. The SEA DB2 informs IMS that it has no knowledge of this thread. IMS does not delete the RRE and it can be displayed by using the IMS DISPLAY OASN command. The SEA DB2 also:
  - Generates message DSN3005 for each IMS RRE for which DB2 has no knowledge.
  - Generates an IFCID 234 trace event.
- When the DB2 subsystems at SJ and LA reconnect with SEA, each detects that the SEA DB2 has cold started. Both the SJ DB2 and the LA DB2:
  - Display message DSNL411.
  - Generate alert A001.
  - Generate an IFCID 204 trace event.
- A display thread report of indoubt threads at both the SJ and LA DB2 subsystems shows the indoubt threads and indicates that the coordinator has cold started.

***System action:*** The DB2 subsystem at both SJ and LA accept the cold start connection from SEA. Processing continues, waiting for a heuristic decision to resolve the indoubt threads.

***System programmer action:*** Call the database administrator.

***Operator action:*** Call the database administrator.

***Database administrator action:*** At this point, neither the SJ nor the LA administrator know if the SEA coordinator was a participant of another coordinator. In this scenario, the SEA DB2 subsystem originated LUWID=16. However, it was a participant for LUWID=15, being coordinated by IMS.

Also not known to the administrator at LA is the fact that SEA distributed the LUWID=16 thread to SJ where it is also indoubt. Likewise, the administrator at SJ does not know that LA has an indoubt thread for the LUWID=16 thread. It is important that both SJ and LA make the same heuristic decision. It is also important that the administrators at SJ and LA determine the originator of the two-phase commit.

The recovery log of the originator indicates whether the decision was commit or abort. The originator might have more accessible functions to determine the decision. Even though the SEA DB2 cold started, you might be able to determine the decision from its recovery log. Or, if the failure occurred before the decision was recorded, you might be able to determine the name of the coordinator, if the SEA DB2 was a participant. A summary report of the SEA DB2 recovery log can be provided by execution of the DSN1LOGP utility.

The LUWID contains the name of the logical unit (LU) where the distributed logical unit of work originated. This logical unit is most likely in the system that originated the two-phase commit.

If an application is distributed, any distributed piece of the application can initiate the two-phase commit. In this type of application, the originator of two-phase commit can be at a different system than that identified by the LUWID. With DB2 private protocol access, the two-phase commit can flow only from the system containing the application that initiates distributed SQL processing. In most cases, this is where the application originates.

The administrator must determine if the LU name contained in the LUWID is the same as the LU name of the SEA DB2 subsystem. If this is not the case (it is the case in this example), then the SEA DB2 is a participant in the logical unit of work, and is being coordinated by a remote system. You must communicate with that system and request that facilities of that system be used to determine if the logical unit of work is to be committed or aborted.

If the LUWID contains the LU name of the SEA DB2 subsystem, then the logical unit of work originated at SEA and is either an IMS, CICS, TSO, or BATCH allied thread of the SEA DB2. The display thread report for indoubt threads at a DB2 participant includes message DSNV458 if the coordinator is remote. This line provides external information provided by the coordinator to assist in identifying the thread. A DB2 coordinator provides the following:

```
connection-name.correlation-id
```

where *connection-name* is:

- SERVER - the thread represents a remote application to the DB2 coordinator and uses DRDA access.
- BATCH - the thread represents a local batch application to the DB2 coordinator.

Anything else represents an IMS or CICS connection name. The thread represents a local application and the commit coordinator is the IMS or CICS system using this connection name.

In our example, the administrator at SJ sees that both indoubt threads have a LUWID with the LU name the same as the SEA DB2 LU name, and furthermore, that one thread (LUWID=15) is an IMS thread and the other thread (LUWID=16) is a batch thread. The LA administrator sees that the LA indoubt thread (LUWID=16) originates at SEA DB2 and is a batch thread.

The originator of a DB2 batch thread is DB2. To determine the commit or abort decision for the LUWID=16 indoubt threads, the SEA DB2 recovery log must be analyzed, if it can be. The DSN1LOGP utility must be executed against the SEA DB2 recovery log, looking for the LUWID=16 entry. There are three possibilities:

1. No entry is found - that portion of the DB2 recovery log was not available.
2. An entry is found but incomplete.
3. An entry is found and the status is committed or aborted.

In the third case, the heuristic decision at SJ and LA for indoubt thread LUWID=16 is indicated by the status indicated in the SEA DB2 recovery log. In the other two cases, the recovery procedure used when cold starting DB2 is important. If recovery was to a previous point in time, then the correct action is to abort. If recovery included repairing the SEA DB2 database, then the SEA administrator might know what decision to make.

The recovery logs at SJ and LA can help determine what activity took place. If it can be determined that updates were performed at either SJ, LA, or both (but not SEA), then if both SJ and LA make the same heuristic action, there should be no data inconsistency. If updates were also performed at SEA, then looking at the SEA data might help determine what action to take. In any case, both SJ and LA should make the same decision.

For the indoubt thread with LUWID=15 (the IMS coordinator), there are several alternative paths to recovery. The SEA DB2 has been restarted. When it reconnects with IMS, message DSN3005 is issued for each thread that IMS is trying to resolve with DB2. The message indicates that DB2 has no knowledge of the thread that is identified by the IMS assigned NID. The outcome for the thread, commit or abort, is included in the message. Trace event IFCID=234 is also written to statistics class 4 containing the same information.

If there is only one such message, or one such entry in statistics class 4, then the decision for indoubt thread LUWID=15 is known and can be communicated to the administrator at SJ. If there are multiple such messages, or multiple such trace events, you must match the IMS NID with the network LUWID. Again, DSN1LOGP should be used to analyze the SEA DB2 recovery log if possible. There are now four possibilities:

1. No entry is found - that portion of the DB2 recovery log was not available.
2. An entry is found but incomplete because of lost recovery log.
3. An entry is found and the status is indoubt.
4. An entry is found and the status is committed or aborted.

In the fourth case, the heuristic decision at SJ for the indoubt thread LUWID=15 is determined by the status indicated in the SEA DB2 recovery log. If an entry is found whose status is indoubt, DSN1LOGP also reports the IMS NID value. The NID is the unique identifier for the logical unit of work in IMS and CICS. Knowing the NID allows correlation to the DSN3005 message, or to the 234 trace event, which provides the correct decision.

If an incomplete entry is found, the NID may or may not have been reported by DSN1LOGP. If it was, use it as previously discussed. If no NID is found, or the SEA DB2 has not been started, or reconnecting to IMS has not occurred, then the *correlation-id* used by IMS to correlate the IMS logical unit of work to the DB2 thread must be used in a search of the IMS recovery log. The SEA DB2 provided this value to the SJ DB2 when distributing the thread to SJ. The SJ DB2 displays this value in the report generated by -DISPLAY THREAD TYPE(INDOUBT).

For IMS, the *correlation-id* is:

```
pst#.psbname
```

In CICS, the *correlation-id* consists of four parts:

```
Byte 1 - Connection Type - G=Group, P=Pool
Byte 2 - Thread Type - T=transaction, G=Group, C=Command
Bytes 3-4 - Thread Number
Bytes 5–8 - Transaction-id
```

## DB2 outage at a server results in a DB2 cold start

***Problem:*** This problem is similar to “DB2 outage at a requester results in a DB2 cold start” on page 469. If the DB2 subsystem at SJ is cold started instead of the DB2 at SEA, then the LA DB2 has the LUWID=15 **2** thread indoubt. The administrator would see that this thread did not originate at SJ, but did originate at SEA. To determine the commit or abort action, the LA administrator would request that -DISPLAY THREAD TYPE(INDOUBT) be issued at the SEA DB2, specifying LUWID=15. IMS would not have any indoubt status for this thread, because it would complete the two-phase commit process with the SEA DB2.

As described in “Communication failure between two systems” on page 467, the DB2 at SEA tells the application that the commit succeeded.

When a participant cold starts, a DB2 coordinator continues to include in the display of indoubt threads all committed threads where the cold starting participant was believed to be indoubt. These entries must be explicitly purged by issuing the RESET INDOUBT command. If a participant has an indoubt thread that cannot be resolved because of coordinator cold start, it can request a display of indoubt threads at the DB2 coordinator to determine the correct action.

## Correcting a heuristic decision

***Problem:*** Assume the conditions of “Communication failure between two systems” on page 467. The LA administrator is called to make a heuristic decision and decides to abort the indoubt thread with LUWID=16. The decision is made without communicating with SEA to determine the proper action. The thread at LA is aborted, while the threads at SEA and SJ are committed. Processing continues at all systems. DB2 at SEA has indoubt resolution responsibility with LA for LUWID=16.

***Symptom:*** When the DB2 at SEA reconnects with the DB2 at LA, indoubt resolution occurs for LUWID=16. Both systems detect heuristic damage and both generate alert A004; each writes an IFCID 207 trace record. Message DSNL400 is displayed at LA and message DSNL403 is displayed at SEA..

***System action:*** Processing continues. Indoubt thread resolution responsibilities have been fulfilled and the thread completes at both SJ and LA.

***System programmer action:*** Call the database administrator.

***Operator action:*** Call the database administrator.

***Database administrator action:*** Correct the damage.

This is not an easy task. Since the time of the heuristic action, the data at LA might have been read or written by many applications. Correcting the damage can involve reversing the effects of these applications as well. The tools available are:

- DSN1LOGP - the summary report of this utility identifies the table spaces modified by the LUWID=16 thread.
- The statistics trace class 4 contains an IFCID 207. entry. This entry identifies the recovery log RBA for the LUWID=16 thread.

Notify the IBM support center about the problem.

# Chapter 23. Recovery from BSDS or log failure during restart

Use this chapter when you have reason to believe that the bootstrap data set (BSDS) or part of the recovery log for DB2 is damaged or lost and that damage is preventing restart. If the problem is discovered at restart, begin with one of these recovery procedures:

“Active log failure” on page 423
“Archive log failure” on page 427
“BSDS failure” on page 429

If the problem persists, return to the procedures in this chapter.

When DB2 recovery log damage terminates restart processing, DB2 issues messages to the console identifying the damage and giving an abend reason code. (The SVC dump title includes a more specific abend reason code to assist in problem diagnosis.) If the explanations in Part 2 of *DB2 Messages and Codes* indicate that restart failed because of some problem not related to a log error, refer to Part 2 of *DB2 Diagnosis Guide and Reference* and contact the IBM support center.

To minimize log problems during restart, the system requires two copies of the BSDS. Dual logging is also recommended.

***Basic approaches to recovery:*** There are two basic approaches to recovery from problems with the log:

- Restart DB2, bypassing the inaccessible portion of the log and rendering some data inconsistent. Then recover the inconsistent objects by using the RECOVER utility, or re-create the data using REPAIR. Methods are described below.
- Restore the entire DB2 subsystem to a prior point of consistency. The method requires that you have first prepared such a point; for suggestions, see “Preparing to recover to a prior point of consistency” on page 383. Methods of recovery are described under “Unresolvable BSDS or log data set problem during restart” on page 494.

***Bypassing the damaged log:*** Even if the log is damaged, and DB2 is started by circumventing the damaged portion, the log is the most important source for determining what work was lost and what data is inconsistent. For information on data sharing considerations, see Chapter 5 of *DB2 Data Sharing: Planning and Administration*.

Bypassing a damaged portion of the log generally proceeds with the following steps:

1. DB2 restart fails. A problem exists on the log, and a message identifies the location of the error. The following abend reason codes, which appear only in the dump title, can be issued for this type of problem. This is not an exhaustive list; other codes might occur.

00D10261
00D10262
00D10263
00D10264
00D10265
00D10266
00D10267
00D10268
00D10329
00D1032A
00D1032B
00D1032C
00E80084

Figure 49 on page 476 illustrates the general problem:

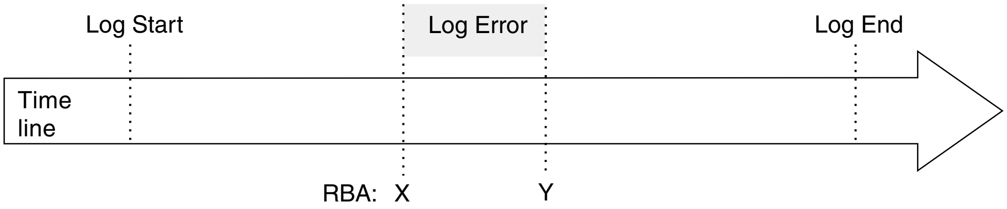


*Figure 49. General problem of damaged DB2 log information*

2. DB2 cannot skip over the damaged portion of the log and continue restart processing. Instead, you restrict processing to only a part of the log that is error free. For example, the damage shown in Figure 49 occurs in the log RBA range from X to Y. You can restrict restart to all of the log before X; then changes later than X are not made. Or you can restrict restart to all of the log after Y; then changes between X and Y are not made. In either case, some amount of data is inconsistent.
3. You identify the data that is made inconsistent by your restart decision. With the SUMMARY option, the DSN1LOGP utility scans the accessible portion of the log and identifies work that must be done at restart, namely, the units of recovery to be completed and the page sets that they modified. (For instructions on using DSN1LOGP, see Part 3 of *DB2 Utility Guide and Reference*.)

   Because a portion of the log is inaccessible, the summary information might not be complete. In some circumstances, your knowledge of work in progress is needed to identify potential inconsistencies.
4. You use the CHANGE LOG INVENTORY utility to identify the portion of the log to be used at restart, and to tell whether to bypass any phase of recovery. You can choose to do a *cold start* and bypass the entire log.
5. You restart DB2. Data that is unaffected by omitted portions of the log is available for immediate access.
6. Before you allow access to any data that is affected by the log damage, you resolve all data inconsistencies. That process is described under “Resolving inconsistencies resulting from conditional restart” on page 500.

***Where to start:*** The specific procedure depends on the phase of restart that was in control when the log problem was detected. On completion, each phase of restart writes a message to the console. You must find the last of those messages in the console log. The next phase after the one identified is the one that was in control when the log problem was detected. Accordingly, start at:
- “Failure during log initialization or current status rebuild” on page 477
- “Failure during forward log recovery” on page 486
- “Failure during backward log recovery” on page 491

As an alternative, determine which, if any, of the following messages was last received and follow the procedure for that message. Other DSN messages can be issued as well.

| Message ID | Procedure to use |
|---|---|
| DSNJ001I | “Failure during log initialization or current status rebuild” on page 477 |
| DSNJ100I | “Unresolvable BSDS or log data set problem during restart” on page 494 |
| DSNJ107I | “Unresolvable BSDS or log data set problem during restart” on page 494 |
| DSNJ1191 | “Unresolvable BSDS or log data set problem during restart” on page 494 |
| DSNR002I | None. Normal restart processing can be expected. |

| Message ID | Procedure to use |
|---|---|
| DSNR004I | “Failure during forward log recovery” on page 486 |
| DSNR005I | “Failure during backward log recovery” on page 491 |
| DSNR006I | None. Normal restart processing can be expected. |
| Other | “Failure during log initialization or current status rebuild” |

Another scenario ( “Failure resulting from total or excessive loss of log data” on page 496) provides information to use if you determine (by using Failure during log initialization or current status rebuild) that an excessive amount (or all) of DB2 log information (BSDS, active, and archive logs) has been lost.

The last scenario in this chapter ( “Resolving inconsistencies resulting from conditional restart” on page 500) can be used to resolve inconsistencies introduced while using one of the restart scenarios in this chapter. If you decide to use “Unresolvable BSDS or log data set problem during restart” on page 494, it is not necessary to use “Resolving inconsistencies resulting from conditional restart” on page 500.

Because of the severity of the situations described, the scenarios identify “Operations Management Action”, rather than “Operator Action”. Operations management might not be performing all the steps in the procedures, but they must be involved in making the decisions about the steps to be performed.

## Failure during log initialization or current status rebuild

***Problem:*** A failure occurred during the log initialization or current status rebuild phase of restart.

***Symptom:*** An abend was issued indicating that restart failed. In addition, the last restart message received was a DSNJ001I message indicating a failure during current status rebuild, or none of the following messages was issued:

DSNJ001I
DSNR004I
DSNR005I

If none of the above messages was issued, the failure occurred during the log initialization phase of restart.

***System action:*** The action depends on whether the failure occurred during log initialization or during current status rebuild.

- **Failure during log initialization:** DB2 terminates because a portion of the log is inaccessible, and DB2 cannot locate the end of the log during restart.
- **Failure during current status rebuild:** DB2 terminates because a portion of the log is inaccessible, and DB2 cannot determine the state of the subsystem (such as outstanding units of recovery, outstanding database writes, or exception database conditions) that existed at the prior DB2 termination.

***Operations management action:*** To correct the problem, choose one of the following approaches:

- Correct the problem that has made the log inaccessible and start DB2 again. To determine if this approach is possible, refer to *DB2 Messages and Codes* for an explanation of the messages and codes received. The explanation will identify the corrective action that can be taken to resolve the problem. In this case, it is not necessary to read the scenarios in this chapter.

- Restore the DB2 log and all data to a prior consistent point and start DB2. This procedure is described in “Unresolvable BSDS or log data set problem during restart” on page 494.
- Start DB2 without completing some database changes. Using a combination of DB2 services and your own knowledge, determine what work will be lost by truncating the log. The procedure for determining the page sets that contain incomplete changes is described in “Restart by truncating the log” on page 479. In order to obtain a better idea of what the problem is, read one of the following sections, depending on when the failure occurred.

## Description of failure during log initialization

Figure 50 illustrates the problem on the log.

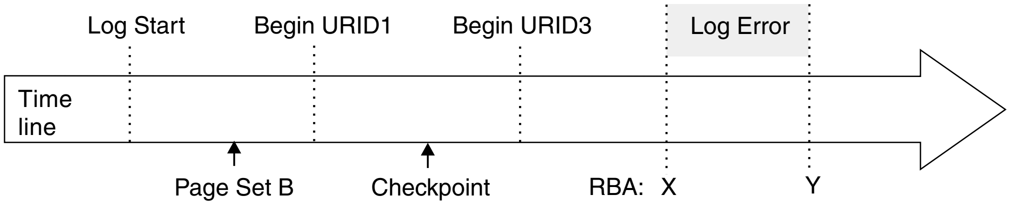


*Figure 50. Failure during log initialization*

The portion of the log between log RBAs X and Y is inaccessible. For failures that occur during the log initialization phase, the following activities occur:

1. DB2 allocates and opens each active log data set that is not in a stopped state.
2. DB2 reads the log until the last log record is located.
3. During this process, a problem with the log is encountered, preventing DB2 from locating the end of the log. DB2 terminates and issues one of the abend reason codes listed in Table 71 on page 480.

During its operations, DB2 periodically records in the BSDS the RBA of the last log record written. This value is displayed in the print log map report as follows:

```
HIGHEST RBA WRITTEN:    00000742989E
```

Because this field is updated frequently in the BSDS, the *highest RBA written* can be interpreted as an approximation of the end of the log. The field is updated in the BSDS when any one of a variety of internal events occurs. In the absence of these internal events, the field is updated each time a complete cycle of log buffers is written. A complete cycle of log buffers occurs when the number of log buffers written equals the value of the OUTPUT BUFFER field of installation panel DSNTIPL. The value in the BSDS is, therefore, relatively close to the end of the log.

To find the actual end of the log at restart, DB2 reads the log forward sequentially, starting at the log RBA that approximates the end of the log and continuing until the actual end of the log is located.

Because the end of the log is inaccessible in this case, some information has been lost. Units of recovery might have successfully committed or modified additional page sets past point X. Additional data might have been written, including those that are identified with writes pending in the accessible portion of the log. New units of recovery might have been created, and these might have modified data. Because of the log error, DB2 cannot perceive these events.

How to restart DB2 is described under “Restart by truncating the log”.

## Description of failure during current status rebuild

Figure 51 illustrates the problem on the log.

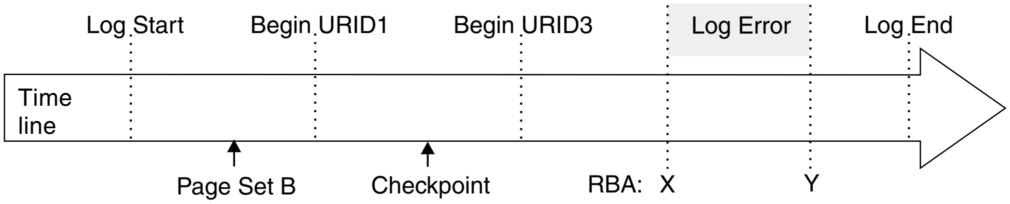


*Figure 51. Failure during current status rebuild*

The portion of the log between log RBAs X and Y is inaccessible. For failures that occur during the current status rebuild phase, the following activities occur:

1. Log initialization completes successfully.
2. DB2 locates the last checkpoint. (The BSDS contains a record of its location on the log.)
3. DB2 reads the log, beginning at the checkpoint and continuing to the end of the log.
4. DB2 reconstructs the subsystem’s state as it existed at the prior termination of DB2.
5. During this process, a problem with the log is encountered, preventing DB2 from reading all required log information. DB2 terminates with one of the abend reason codes listed in Table 71 on page 480.

Because the end of the log is inaccessible in this case, some information has been lost. Units of recovery might have successfully committed or modified additional page sets past point X. Additional data might have been written, including those that are identified with writes pending in the accessible portion of the log. New units of recovery might have been created, and these might have modified data. Because of the log error, DB2 cannot perceive these events.

How to restart DB2 is described under “Restart by truncating the log”.

## Restart by truncating the log

When a portion of the log is inaccessible, during the log initialization or current status rebuild phases of restart, DB2 cannot identify precisely what units of recovery failed to complete, what page sets those modified, and what page sets have writes pending. This procedure tells how to gather that information and restart.

### Step 1: Find the log RBA after the inaccessible part of the log

The log damage is illustrated in Figure 50 on page 478 and in Figure 51. The range of the log between RBAs X and Y is inaccessible to all DB2 processes.

Use the abend reason code accompanying the X'04E' abend and the message on the title of the accompanying dump at the operator’s console, to find the name and page number of a procedure in Table 71 on page 480. Use that procedure to find X and Y.

*Table 71. Abend reason codes and messages*

| Abend Reason Code | Message | Procedure Name and Page | General Error Description |
|---|---|---|---|
| 00D10261<br>00D10262<br>00D10263<br>00D10264<br>00D10265<br>00D10266<br>00D10267<br>00D10268 | DSNJ012I | RBA 1, page 480 | Log record is logically damaged |
| 00D10329 | DSNJ106I | RBA 2, page 480 | I/O error occurred while log record was being read |
| 00D1032A | DSNJ113E | RBA 3, page 481 | Log RBA could not be found in BSDS |
| 00D1032B | DSNJ103I | RBA 4, page 481 | Allocation error occurred for an archive log data set |
| 00D1032B | DSNJ007I | RBA 5, page 482 | The operator canceled a request for archive mount |
| 00D1032C | DSNJ104I | RBA 4, page 481 | Open error occurred for an archive and active log data set |
| 00E80084 | DSNJ103I | RBA 4, page 481 | Active log data set named in the BSDS could not be allocated during log initialization |

***Procedure RBA 1:*** The message accompanying the abend identifies the log RBA of the first inaccessible log record that DB2 detects. For example, the following message indicates a logical error in the log record at log RBA X'7429ABA'.

```
DSNJ012I ERROR D10265 READING RBA 000007429ABA
         IN DATA SET DSNCAT.LOGCOPY2.DS01
         CONNECTION-ID=DSN,
         CORRELATION-ID=DSN
```

Figure 138 on page 963 shows that a given physical log record is actually a set of logical log records (the log records generally spoken of) and the log control interval definition (LCID). DB2 stores logical records in blocks of physical records to improve efficiency. When this type of an error on the log occurs during log initialization or current status rebuild, all log records within the physical log record are inaccessible. Therefore, the value of X is the log RBA that was reported in the message rounded down to a 4 KB boundary (X'7429000').

Continue with step 2 on page 482.

***Procedure RBA 2:*** The message accompanying the abend identifies the log RBA of the first inaccessible log record that DB2 detects. For example, the following message indicates an I/O error in the log at RBA X'7429ABA'.

```
DSNJ106I LOG READ ERROR DSNAME=DSNCAT.LOGCOPY2.DS01,
         LOGRBA=000007429ABA,ERROR STATUS=0108320C
```

Figure 138 on page 963 shows that a given physical log record is actually a set of logical log records (the log records generally spoken of) and the LCID. When this type of an error on the log occurs during log initialization or current status rebuild, all log records within the physical log record and beyond it to the end of the log data set are inaccessible to the log initialization or current status rebuild phase of restart. Therefore, the value of X is the log RBA that was reported in the message, rounded down to a 4 KB boundary (X'7429000').

Continue with step 2 on page 482.

***Procedure RBA 3:*** The message accompanying the abend identifies the log RBA of the inaccessible log record. This log RBA is not registered in the BSDS.

For example, the following message indicates that the log RBA X'7429ABA' is not registered in the BSDS:

```
DSNJ113E RBA 000007429ABA NOT IN ANY ACTIVE OR ARCHIVE
        LOG DATA SET.  CONNECTION-ID=DSN, CORRELATION-ID=DSN
```

The print log map utility can be used to list the contents of the BSDS. For an example of the output, see the description of print log map (DSNJU004) in Part 3 of *DB2 Utility Guide and Reference*.

Figure 138 on page 963 shows that a given physical log record is actually a set of logical log records (the log records generally spoken of) and the LCID. When this type of an error on the log occurs during log initialization or current status rebuild, all log records within the physical log record are inaccessible.

Using the print log map output, locate the RBA closest to, but less than, X'7429ABA' for the value of X. If there is not an RBA that is less than X'7429ABA', a considerable amount of log information has been lost. If this is the case, continue with “Failure resulting from total or excessive loss of log data” on page 496.

If there is a value for X, continue with step 2 on page 482.

***Procedure RBA 4:*** The message accompanying the abend identifies an entire data set that is inaccessible. For example, the following message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible, and the STATUS field identifies the code that is associated with the reason for the data set being inaccessible. For an explanation of the STATUS codes, see the explanation for the message in Part 2 of *DB2 Messages and Codes* .

```
DSNJ103I - csect-name LOG ALLOCATION ERROR
        DSNAME=DSNCAT.ARCHLOG1.A0000009,ERROR
        STATUS=04980004
        SMS REASON CODE=00000000
```

To determine the value of X, run the print log map utility to list the log inventory information. For an example of the output, see the description of print log map (DSNJU004) in Part 3 of *DB2 Utility Guide and Reference*. The output provides each log data set name and its associated log RBA range—the values of X and Y.

Verify the accuracy of the information in the print log map utility output for the active log data set with the lowest RBA range. For this active log data set only, the information in the BSDS is potentially inaccurate for the following reasons:

- When an active log data set is full, archiving is started. DB2 then selects another active log data set, usually the data set with the lowest RBA. This selection is made so that units of recovery do not have to wait for the archive operation to complete before logging can continue. However, if a data set has not been archived, nothing beyond it has been archived, and the procedure is ended.
- When logging has begun on a reusable data set, DB2 updates the BSDS with the new log RBA range for the active log data set, and marks it as *Not Reusable*. The process of writing the new information to the BSDS can be delayed by other

processing. It is therefore possible for a failure to occur between the time that logging to a new active log data set begins and the time that the BSDS is updated. In this case, the BSDS information is not correct.

The log RBA that appears for the active log data set with the lowest RBA range in the print log map utility output is valid, provided that the data set is marked *Not Reusable*. If the data set is marked *Reusable*, it can be assumed for the purposes of this restart that the starting log RBA (X) for this data set is one greater than the highest log RBA listed in the BSDS for all other active log data sets.

Continue with step 2 on page 482.

***Procedure RBA 5:*** The message accompanying the abend identifies an entire data set that is inaccessible. For example, the following message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible. The operator canceled a request for archive mount, resulting in the following message:

```
DSNJ007I OPERATOR CANCELED MOUNT OF ARCHIVE
         DSNCAT.ARCHLOG1.A0000009 VOLSER=5B225.
```

To determine the value of X, run the print log map utility to list the log inventory information. For an example of the output, see the description of print log map (DSNJU004) in Part 3 of *DB2 Utility Guide and Reference*. The output provides each log data set name and its associated log RBA range: the values of X and Y.

Continue with step 2 on 482.

## Step 2: Identify lost work and inconsistent data

1. Obtain available information to help you determine the extent of the loss.

   It is impossible for DB2 to determine what units of recovery are not completed, what database state information is lost, or what data is inconsistent in this situation. The log contains all such information, but the information is not available. The following steps explain what to do to obtain the information that is available within DB2 to help determine the extent of the loss. The steps also explain how to start DB2 in this situation.

   After restart, data is inconsistent. Results of queries and any other operations on such data vary from incorrect results to abends. Abends that occur either identify an inconsistency in the data or incorrectly assume the existence of a problem in the DB2 internal algorithms. **If the inconsistent page sets cannot be identified and the problems in them cannot be resolved after starting DB2, there is a risk in following this procedure and allowing access to inconsistent data**.

   a. Execute the print log map utility. The report it produces includes a description of the last 100 checkpoints and provides, for each checkpoint:

      The location in the log of the checkpoint (begin and end RBA)
      The date and time of day that the checkpoint was performed.

   b. Locate the checkpoint on the log prior to the point of failure (X). Do that by finding the first checkpoint with an end RBA that is less than X.

      If you cannot find such a checkpoint, this means that a considerable amount of log has been lost. In this case, either follow the procedure under “Failure resulting from total or excessive loss of log data” on page 496 or the procedure under “Unresolvable BSDS or log data set problem during restart” on page 494.

      If the checkpoint is found, look at the date and time it was performed. If the checkpoint is several days old (and DB2 was operational during the interim), either follow the procedure under “Failure resulting from total or excessive loss of log data” on page 496 or the procedure under “Unresolvable BSDS or log data set problem during restart” on page 494.

Otherwise, continue with the next step.

2. Determine what work is lost and what data is inconsistent.

   The portion of the log representing activity that occurred before the failure provides information about work that was in progress at that point. From this information, it might be possible to deduce the work that was in progress within the inaccessible portion of the log. If use of DB2 was limited at the time or if DB2 was dedicated to a small number of activities (such as batch jobs performing database loads or image copies), it might be possible to accurately identify the page sets that were made inconsistent. To make the identification, extract a summary of the log activity up to the point of damage in the log by using the DSN1LOGP utility described in Part 3 of *DB2 Utility Guide and Reference*.

   Use the DSN1LOGP utility to specify the “BEGIN CHECKPOINT” RBA prior to the point of failure, which was determined in the previous step as the RBASTART. End the DSN1LOGP scan prior to the point of failure on the log (X - 1) by using the RBAEND specification.

   Specifying the last complete checkpoint is very important for ensuring that complete information is obtained from DSN1LOGP.

   Specify the SUMMARY(ONLY) option to produce a summary report.

   Figure 52 is an example of a DSN1LOGP job to obtain summary information for the checkpoint discussed previously.

```
//ONE EXEC PGM=DSN1LOGP
//STEPLIB  DD DSN=prefix.SDSNLOAD,DISP=SHR
//SYSABEND DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSSUMRY DD SYSOUT=A
//BSDS     DD DSN=DSNCAT.BSDS01,DISP=SHR
//SYSIN    DD *
   RBASTART (7425468) RBAEND (7428FFF) SUMMARY (ONLY)
/*
```

*Figure 52. Sample JCL for obtaining DSN1LOGP summary output for restart*

3. Analyze the DSN1LOGP utility output.

   The summary report that is placed in the SYSSUMRY file includes two sections of information: a summary of completed events (not shown here) and a restart summary shown in Figure 53 on page 484. Following this figure is a description of the sample output.

```
DSN1157I RESTART SUMMARY

DSN1153I DSN1LSIT CHECKPOINT
               STARTRBA=000007425468  ENDRBA=000007426C6C  STARTLRSN=AA527AA809DF  ENDLRSN=AA527AA829F4
               DATE=92.284  TIME=14:49:25

DSN1162I DSN1LPRT  UR  CONNID=BATCH     CORRID=PROGRAM2      AUTHID=ADMF001   PLAN=TCEU02
             START DATE=92.284 TIME=11:12:01  DISP=INFLIGHT    INFO=COMPLETE
             STARTRBA=0000063DA17B  STARTLRSN=A974FAFF27FF  NID=*
             LUWID=DB2NET.LUND0.A974FAFE6E77.0001  COORDINATOR=*
             PARTICIPANTS=*

             DATA MODIFIED:
                  DATABASE=0101=STVDB02   PAGESET=0002=STVTS02

DSN1162I DSN1LPRT  UR  CONNID=BATCH     CORRID=PROGRAM5      AUTHID=ADMF001   PLAN=TCEU02
             START DATE=92.284 TIME=11:21:02  DISP=INFLIGHT    INFO=COMPLETE
             STARTRBA=000006A57C57  STARTLRSN=A974FAFF2801  NID=*
             LUWID=DB2NET.LUND0.A974FAFE6FFF.0003  COORDINATOR=*
             PARTICIPANTS=*

             DATA MODIFIED:
                  DATABASE=0104=STVDB05   PAGESET=0002=STVTS05

DSN1162I DSN1LPRT  UR  CONNID=TEST0001  CORRID=CTHDCORID001  AUTHID=MULT002   PLAN=DONSQL1
             START DATE=92.278 TIME=06:49:33  DISP=INDOUBT     INFO=PARTIAL
             STARTRBA=000005FBCC4F  STARTLRSN=A974FBAF2302  NID=*
             LUWID=DB2NET.LUND0.B978FAFEFAB1.0000  COORDINATOR=*
             PARTICIPANTS=*

             NO DATA MODIFIED (BASED ON INCOMPLETE LOG INFORMATION)

DSN1162I UR  CONNID=BATCH     CORRID=PROGRAM2      AUTHID=ADMF001   PLAN=TCEU02
             START DATE=92.284 TIME=11:12:01  DISP=INFLIGHT    INFO=COMPLETE
             START=0000063DA17B

DSN1160I DATABASE WRITES PENDING:
          DATABASE=0001=DSNDB01    PAGESET=004F=SYSUTIL    START=000007425468
          DATABASE=0102   PAGESET=0015   START=000007425468
```

*Figure 53. Partial sample of DSN1LOGP summary output*

The heading message:

```
DSN1157I RESTART SUMMARY
```

is followed by messages that identify the units of recovery that have not yet completed and the page sets that they modified.

Following the summary of outstanding units of recovery is a summary of page sets with database writes pending.

In each case (units of recovery or databases with pending writes), the earliest required log record is identified by the START information. In this context, START information is the log RBA of the earliest log record required in order to complete outstanding writes for this page set.

Those units of recovery with a START log RBA equal to, or prior to, the point Y cannot be completed at restart. All page sets modified by such units of recovery are inconsistent after completion of restart using this procedure.

All page sets identified in message DSN1160I with a START log RBA value equal to, or prior to, the point Y have database changes that cannot be written to disk. As in the case previously described, all such page sets are inconsistent after completion of restart using this procedure.

At this point, it is only necessary to identify the page sets in preparation for restart. After restart, the problems in the page sets that are inconsistent must be resolved.

Because the end of the log is inaccessible, some information has been lost, therefore, the information is inaccurate. Some of the units of recovery that appear to be inflight might have successfully committed, or they could have modified additional page sets beyond point X. Additional data could have been written, including those page sets that are identified as having writes pending in the accessible portion of the log. New units of recovery could have been created, and these can have modified data. DB2 cannot detect that these events occurred.

From this and other information (such as system accounting information and console messages), it could be possible to determine what work was actually outstanding and which page sets will be inconsistent after starting DB2, because the record of each event contains the date and time to help determine how recent the information is. In addition, the information is displayed in chronological sequence.

## Step 3: Determine what status information has been lost

Some amount of system status information might have been lost. In some cases, you will know what information has been lost (such as the case in which utilities are in progress). In other cases, messages about the loss of status information (such as in the cases of deferred restart pending or write error ranges) might be received. If system status information has been lost, it could be possible to reconstruct this information from recent console displays, messages, and abends that alerted you to these conditions. The page sets that are in such a state must be identified because they are inconsistent and inconsistencies must be resolved.

## Step 4: Truncate the log at the point of error

No DB2 process, including RECOVER, allows a gap in the log RBA sequence. You cannot process up to point X, skip over points X through Y, and continue after Y.

Use the change log inventory utility to create a conditional restart control record (CRCR) in the BSDS, identifying the end of the log (X) to use on a subsequent restart. The value is the RBA at which DB2 begins writing new log records. If point X is X'7429000', on the CRESTART control statement, specify ENDRBA=7429000.

At restart, DB2 discards the portion of the log beyond X'7429000' before processing the log for completing work (such as units of recovery and database writes). Unless otherwise directed, normal restart processing is performed within the scope of the log. Because log information has been lost, DB2 errors can occur. For example, a unit of recovery that has actually been committed can be rolled back. Also, some changes made by that unit of recovery might not be rolled back because information about data changes has been lost.

To minimize such errors, use this change log inventory control statement:

```
CRESTART  CREATE,ENDRBA=7429000,FORWARD=NO,BACKOUT=NO
```

When DB2 is started (in Step 6), it:

1. Discards from the checkpoint queue any entries with RBAs beyond the ENDRBA value in the CRCR (X'7429000' in the previous example).

2. Reconstructs the system status up to the point of log truncation.
3. Completes all database writes that are identified by the DSN1LOGP summary report and have not already been performed.
4. Completes all units of recovery that have committed or are indoubt. The processing varies for different unit of recovery states as described in “Normal restart and recovery” on page 348.
5. Does not back out inflight or in-abort units of recovery. Inflight units of recovery might have been committed. Data modified by in-abort units of recovery could have been modified again after the point of damage on the log. Thus, inconsistent data can be left in tables modified by inflight or indoubt URs. Backing out without the lost log information might introduce further inconsistencies.

### Step 5: Start DB2

At the end of restart, the conditional restart control record (CRCR) is marked *DEACTIVATED* to prevent its use on a later restart. Until the restart has completed successfully, the CRCR is in effect. Start DB2 with ACCESS (MAINT) until data is consistent or page sets are stopped.

### Step 6: Resolve data inconsistency problems

After successfully restarting DB2, resolve all data inconsistency problems as described in “Resolving inconsistencies resulting from conditional restart” on page 500.

## Failure during forward log recovery

***Problem:*** A failure occurred during the forward log recovery phase of restart.

***Symptom:*** An abend was issued, indicating that restart had failed. In addition, the last restart message received was a DSNR004I message indicating that log initialization had completed and thus the failure occurred during forward log recovery.

***System action:*** DB2 terminates because a portion of the log is inaccessible, and DB2 is therefore unable to guarantee the consistency of the data after restart.

***Operations management action:*** To start DB2 successfully, choose one of the following approaches:

- Correct the problem that has made the log inaccessible and start DB2 again. To determine if this approach is possible, refer to *DB2 Messages and Codes* for an explanation of the messages and codes received. The explanation will identify any corrective action that can be taken to resolve the problem. In this case, it is not necessary to read the scenarios in this chapter.
- Restore the DB2 log and all data to a prior consistent point and start DB2. This procedure is described in “Unresolvable BSDS or log data set problem during restart” on page 494.
- Start DB2 without completing some database changes. The exact changes cannot be identified; all that can be determined is which page sets might have incomplete changes. The procedure for determining which page sets contain incomplete changes is described in “Starting DB2 by limiting restart processing” on page 487. Continue reading this chapter to obtain a better idea of what the problem is.

Figure 54 illustrates the problem on the log.

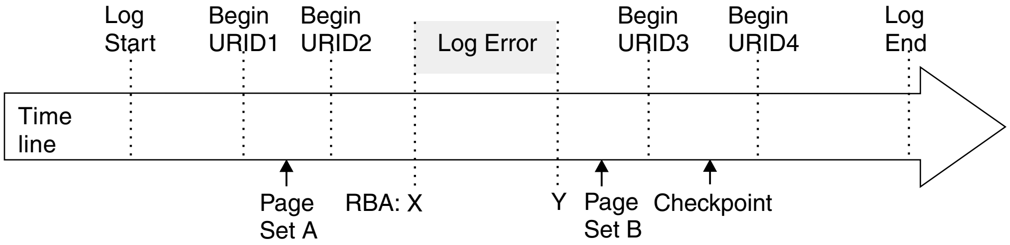


*Figure 54. Illustration of failure during forward log recovery*

The portion of the log between log RBA X and Y is inaccessible. The log initialization and current status rebuild phases of restart completed successfully. Restart processing was reading the log in a forward direction beginning at some point prior to X and continuing to the end of the log. Because of the inaccessibility of log data (between points X and Y), restart processing cannot guarantee the completion of any work that was outstanding at restart prior to point Y.

For purposes of discussion, assume the following work was outstanding at restart:

- The unit of recovery identified as URID1 was in-commit.
- The unit of recovery identified as URID2 was inflight.
- The unit of recovery identified as URID3 was in-commit.
- The unit of recovery identified as URID4 was inflight.
- Page set A had writes pending prior to the error on the log, continuing to the end of the log.
- Page set B had writes pending after the error on the log, continuing to the end of the log.

The earliest log record for each unit of recovery is identified on the log line in Figure 54. In order for DB2 to complete each unit of recovery, DB2 requires access to all log records from the beginning point for each unit of recovery to the end of the log.

The error on the log prevents DB2 from guaranteeing the completion of any outstanding work that began prior to point Y on the log. Consequently, database changes made by URID1 and URID2 might not be fully committed or backed out. Writes pending for page set A (from points in the log prior to Y) will be lost.

## Starting DB2 by limiting restart processing

This procedure describes how to start DB2 when a portion of the log is inaccessible during forward recovery. It also describes how to identify the units of recovery for which database changes cannot be fully guaranteed (either committed or backed out) and the page sets that these units of recovery changed. You must determine which page sets are involved because after this procedure is used, the page sets will contain inconsistencies that must be resolved. In addition, using this procedure results in the completion of all database writes that are pending. For a description of this process of writing database pages to disk, see “Tuning database buffer pools” on page 549.

### Step 1: Find the log RBA after the inaccessible part of the log

The log damage is shown in Figure 54. The range of the log between RBA X and RBA Y is inaccessible to all DB2 processes.

Use the abend reason code accompanying the X'04E' abend, and the message on the title of the accompanying dump at the operator's console, to find the name and page number of a procedure in Table 72. Use that procedure to find X and Y.

*Table 72. Abend reason codes and messages*

| Abend Reason Code | Message | Procedure Name and Page | General Error Description |
|---|---|---|---|
| 00D10261<br>00D10262<br>00D10263<br>00D10264<br>00D10265<br>00D10266<br>00D10267<br>00D10268 | DSNJ012I | RBA 1, page 488 | Log record is logically damaged |
| 00D10329 | DSNJ106I | RBA 2, page 488 | I/O error occurred while log record was being read |
| 00D1032A | DSNJ113E | RBA 3, page 489 | Log RBA could not be found in BSDS |
| 00D1032B | DSNJ103I | RBA 4, page 489 | Allocation error occurred for an archive log data set |
| 00D1032B | DSNJ007I | RBA 5, page 490 | The operator canceled a request for archive mount |
| 00D1032C | DSNJ104E | RBA 4, page 489 | Open error occurred for an archive log data set |
| 00E80084 | DSNJ103I | RBA 4, page 489 | Active log data set named in the BSDS could not be allocated during log initialization. |

***Procedure RBA 1:*** The message accompanying the abend identifies the log RBA of the first inaccessible log record that DB2 detects. For example, the following message indicates a logical error in the log record at log RBA X'7429ABA':

```
DSNJ012I ERROR D10265 READING RBA 000007429ABA
         IN DATA SET DSNCAT.LOGCOPY2.DS01
         CONNECTION-ID=DSN
         CORRELATION-ID=DSN
```

Figure 138 on page 963 shows that a given physical log record is actually a set of logical log records (the log records generally spoken of) and the log control interval definition (LCID). When this type of an error on the log occurs during forward log recovery, all log records within the physical log record, as described, are inaccessible. Therefore, the value of X is the log RBA that was reported in the message, rounded down to a 4K boundary (that is, X'7429000').

For purposes of following the steps in this procedure, assume that the extent of damage is limited to the single physical log record. Therefore, calculate the value of Y as the log RBA that was reported in the message, rounded up to the end of the 4K boundary (that is, X'7429FFF').

Continue with step 2 on page 490.

***Procedure RBA 2:*** The message accompanying the abend identifies the log RBA of the first inaccessible log record that DB2 detects. For example, the following message indicates an I/O error in the log at RBA X'7429ABA':

```
DSNJ106I LOG READ ERROR DSNAME=DSNCAT.LOGCOPY2.DS01,
         LOGRBA=000007429ABA, ERROR STATUS=0108320C
```

Figure 138 on page 963 shows that a given physical log record is actually a set of logical log records (the log records generally spoken of) and the LCID. When this type of an error on the log occurs during forward log recovery, all log records within the physical log record and beyond it to the end of the log data set are inaccessible to the forward recovery phase of restart. Therefore, the value of X is the log RBA that was reported in the message, rounded down to a 4K boundary (that is, X'7429000').

To determine the value of Y, run the print log map utility to list the log inventory information. For an example of this output, see the description of print log map (DSNJU004) in Part 3 of *DB2 Utility Guide and Reference*. Locate the data set name and its associated log RBA range. The RBA of the end of the range is the value Y.

Continue with step 2 on page 490.

***Procedure RBA 3:*** The message accompanying the abend identifies the log RBA of the inaccessible log record. This log RBA is not registered in the BSDS.

For example, the following message indicates that the log RBA X'7429ABA' is not registered in the BSDS:

```
DSNJ113E RBA 000007429ABA NOT IN ANY ACTIVE OR ARCHIVE
        LOG DATA SET.  CONNECTION-ID=DSN, CORRELATION-ID=DSN
```

Use the print log map utility to list the contents of the BSDS. For an example of this output, see the description of print log map (DSNJU004) in Part 3 of *DB2 Utility Guide and Reference*.

Figure 138 on page 963 shows that a given physical log record is actually a set of logical log records (the log records generally spoken of) and the LCID. When this type of error on the log occurs during forward log recovery, all log records within the physical log record are inaccessible.

Using the print log map output, locate the RBA closest to, but less than, X'7429ABA'. This is the value of X. If an RBA less than X'7429ABA' cannot be found, the value of X is zero. Locate the RBA closest to, but greater than, X'7429ABA'. This is the value of Y.

Continue with step 2 on page 490.

***Procedure RBA 4:*** The message accompanying the abend identifies an entire data set that is inaccessible. For example, the following message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible. The STATUS field identifies the code that is associated with the reason for the data set being inaccessible. For an explanation of the STATUS codes, see the explanation for the message in *DB2 Messages and Codes* .

```
DSNJ103I LOG ALLOCATION ERROR
        DSNAME=DSNCAT.ARCHLOG1.A0000009, ERROR
        STATUS=04980004
        SMS REASON CODE=00000000
```

To determine the values of X and Y, run the print log map utility to list the log inventory information. For an example of this output, see the description of print log map (DSNJU004) in Part 2 of *DB2 Utility Guide and Reference*. The output provides each log data set name and its associated log RBA range: the values of X and Y.

Continue with step 2 on page 490.

***Procedure RBA 5:*** The message accompanying the abend identifies an entire data set that is inaccessible. For example, the following message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible. The operator canceled a request for archive mount resulting in the following message:

```
DSNJ007I OPERATOR CANCELED MOUNT OF ARCHIVE
        DSNCAT.ARCHLOG1.A0000009 VOLSER=5B225.
```

To determine the values of X and Y, run the print log map utility to list the log inventory information. For an example of the output, see the description of print log map (DSNJU004) in Part 3 of *DB2 Utility Guide and Reference*. The output provides each log data set name and its associated log RBA range: the values of X and Y. Continue with Step 2 on page 490.

## Step 2: Identify incomplete units of recovery and inconsistent page sets

Units of recovery that cannot be fully processed are considered *incomplete units of recovery*. Page sets that will be inconsistent following completion of restart are considered *inconsistent page sets*. Take the following steps to identify them:

1. Determine the location of the latest checkpoint on the log. Determine this by looking at one of the following sources, whichever is more convenient:
   - The operator’s console contains the following message, identifying the location of the start of the last checkpoint on the log at log RBA X'876B355'.

     ```
     DSNR003I RESTART ... PRIOR CHECKPOINT
                          RBA=00007425468
     ```

   - The print log map utility output identifies the last checkpoint, including its BEGIN CHECKPOINT RBA.
2. Run the DSN1LOGP utility to obtain a report of the outstanding work that is to be completed at the next restart of DB2. When you run the DSN1LOGP utility, specify the checkpoint RBA as the STARTRBA and the SUMMARY(ONLY) option. It is very important that you include the last complete checkpoint from running DSN1LOGP in order to obtain complete information.

   Figure 52 on page 483 shows an example of the DSN1LOGP job submitted for the checkpoint that was reported in the DSNR003I message.

Analyze the output of the DSN1LOGP utility. The summary report that is placed in the SYSSUMRY file contains two sections of information. For an example of SUMMARY output, see Figure 53 on page 484; and for an example of the program that results in the output, see Figure 52 on page 483.

## Step 3: Restrict restart processing to the part of the log after the damage

Use the change log inventory utility to create a conditional restart control record (CRCR) in the BSDS. Identify the accessible portion of the log beyond the damage by using the STARTRBA specification, which will be used at the next restart. Specify the value Y+1 (that is, if Y is X'7429FFF', specify STARTRBA=742A000). Restart will restrict its processing to the portion of the log beginning with the specified STARTRBA and continuing to the end of the log. A sample change log inventory utility control statement is:

```
CRESTART CREATE,STARTRBA=742A000
```

### Step 4: Start DB2

At the end of restart, the CRCR is marked *DEACTIVATED* to prevent its use on a subsequent restart. Until the restart is complete, the CRCR will be in effect. Use -START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped.

### Step 5: Resolve inconsistent data problems

Following the successful start of DB2, all data inconsistency problems must be resolved. “Resolving inconsistencies resulting from conditional restart” on page 500 describes how to do this. At this time, all other data can be made available for use.

## Failure during backward log recovery

***Problem:*** A failure occurred during the backward log recovery phase of restart.

***Symptom:*** An abend was issued that indicated that restart failed because of a log problem. In addition, the last restart message received was a DSNR005I message, indicating that forward log recovery completed and thus the failure occurred during backward log recovery.

***System action:*** DB2 terminates because a portion of the log that it needs is inaccessible, and DB2 is therefore unable to rollback some database changes during restart.

***Operations management action:*** To start DB2, choose one of the following approaches:

1. Correct the problem that has made the log inaccessible and start DB2 again. To determine whether this approach is possible, refer to *DB2 Messages and Codes* for an explanation of the messages and codes received. The explanation identifies the corrective action to take to resolve the problem. In this case, it is not necessary to read the scenarios in this chapter.
2. Restore the DB2 log and all data to a prior consistent point and start DB2. This procedure is described in “Unresolvable BSDS or log data set problem during restart” on page 494.
3. Start DB2 without rolling back some database changes. The exact database changes cannot be identified. All that can be determined is which page sets contain incomplete changes and which units of recovery made modifications to those page sets. The procedure for determining which page sets contain incomplete changes and which units of recovery made the modifications is described in “Bypassing backout before restarting” on page 492. Continue reading this chapter to obtain a better idea of how to fix the problem.

Figure 55 illustrates the problem on the log.

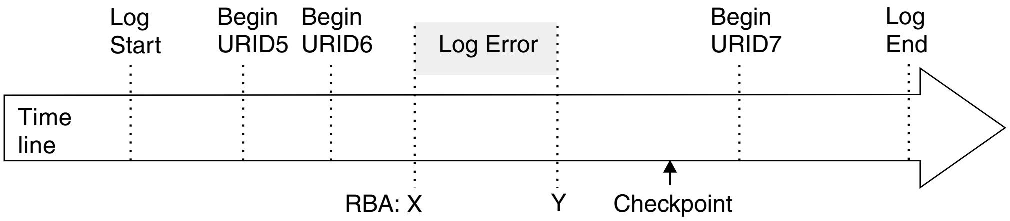


*Figure 55. Illustration of failure during backward log recovery*

The portion of the log between log RBA X and Y is inaccessible. Restart was reading the log in a backward direction beginning at the end of the log and continuing backward to the point marked by *Begin URID5* in order to back out the changes made by URID5, URID6, and URID7. You can assume that DB2 determined that these units of recovery were inflight or in-abort. The portion of the log from point Y to the end has been processed. However, the portion of the log from *Begin URID5* to point Y has not been processed and cannot be processed by restart. Consequently, database changes made by URID5 and URID6 might not be fully backed out. All database changes made by URID7 have been fully backed out, but these database changes might not have been written to disk. A subsequent restart of DB2 causes these changes to be written to disk during forward recovery.

## Bypassing backout before restarting

This procedure describes how to start DB2 when a portion of the log is inaccessible during backward recovery. It also describes how to identify the units of recovery that cannot be fully backed out and the page sets that are inconsistent because they were changed by the units of recovery that did not complete.

1. Determine the units of recovery that cannot be backed out and the page sets that will be inconsistent following completion of restart. To do this, take the following steps:
   a. Determine the location of the latest checkpoint on the log. This can be determined by looking at one of the following sources, whichever is more convenient:
      - The operator’s console contains message DSNR003I, which identifies the location of the start of the last checkpoint on the log at log RBA X'7425468'.

        ```
        DSNR003I RESTART ... PRIOR CHECKPOINT
                             RBA=00007425468
        ```
      - Print log map utility output identifies the last checkpoint, including its BEGIN CHECKPOINT RBA.

   b. Execute the DSN1LOGP utility to obtain a report of the outstanding work that is to be completed at the next restart of DB2. When you run DSN1LOGP, specify the checkpoint RBA as the RBASTART and the SUMMARY(ONLY) option. Include the last complete checkpoint in the execution of DSN1LOGP in order to obtain complete information.

      Figure 53 on page 484 shows an example of the DSN1LOGP job submitted for the checkpoint that was reported in the DSNR003I message.

   Analyze the output of the DSN1LOGP utility. The summary report that is placed in the SYSSUMRY file contains two sections of information. The sample report output shown in Figure 53 on page 484 resulted from the invocation shown in Figure 52 on page 483. The following description refers to that sample output.

   The first section is headed by the following message:

   ```
   DSN1150I SUMMARY OF COMPLETED EVENTS
   ```

   That message is followed by others that identify completed events, such as completed units of recovery. That section does not apply to this procedure.

   The second section is headed by this message:

   ```
   DSN1157I RESTART SUMMARY
   ```

   That message is followed by others that identify units of recovery that are not yet completed and the page sets that they modified. An example of the DSN1162I messages is shown in Figure 53 on page 484.

Following the summary of outstanding units of recovery is a summary of page sets with database writes pending. An example of the DSN1160I message is shown in Figure 53 on page 484.

The restart processing that failed was able to complete all units of recovery processing within the accessible scope of the log following point Y. Database writes for these units of recovery are completed during the forward recovery phase of restart on the next restart. Therefore, do not bypass the forward recovery phase. All units of recovery that can be backed out have been backed out.

All remaining units of recovery to be backed out (DISP=INFLIGHT or DISP=IN-ABORT) are bypassed on the next restart because their STARTRBA values are less than the RBA of point Y. Therefore, all page sets modified by those units of recovery are inconsistent following restart. This means that some changes to data might not be backed out. At this point, it is only necessary to identify the page sets in preparation for restart.

2. Direct restart to bypass backward recovery processing. Use the change log inventory utility to create a conditional restart control record (CRCR) in the BSDS. Direct restart to bypass backward recovery processing during the subsequent restart by using the BACKOUT specification. At restart, all units of recovery requiring backout are declared complete by DB2, and log records are generated to note the end of the unit of recovery. The change log inventory utility control statement is:

   ```
   CRESTART CREATE,BACKOUT=NO
   ```

3. Start DB2. At the end of restart, the CRCR is marked *DEACTIVATED* to prevent its use on a subsequent restart. Until the restart is complete, the CRCR is in effect. Use START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped.
4. Resolve all inconsistent data problems. Following the successful start of DB2, all data inconsistency problems must be resolved. “Resolving inconsistencies resulting from conditional restart” on page 500 describes how to do this. At this time, all other data can be made available for use.

## Failure during a log RBA read request

***Problem:*** The BSDS is wrapping around too frequently when log RBA read requests are submitted; when the last archive log data sets were added to the BSDS, the maximum allowable number of log data sets in the BSDS was exceeded. This caused the earliest data sets in the BSDS to be displaced by the new entry. Subsequently, the requested RBA containing the dropped log data set cannot be read after the wrap occurs.

***Symptom:*** Abend code 00D1032A and message DSNJ113E are displayed:

```
DSNJ113E RBA log-rba NOT IN ANY ACTIVE OR ARCHIVE
        LOG DATA SET.  CONNECTION-ID=aaaaaaaa, CORRELATION-ID=aaaaaaaa
```

***System programmer action:***

1. Stop DB2 with the -STOP DB2 command, if it has not already been stopped automatically as a result of the problem.
2. Check any other messages and reason codes displayed and correct the errors indicated. Locate the output from an old print log map run, and identify the data

set that contains the missing RBA. If the data set has not been reused, run the change log inventory utility to add this data set back into the inventory of log data sets.

3. Increase the maximum number of archive log volumes that can be recorded in the BSDS. To do this, update the MAXARCH system parameter value as follows:
   a. Start the installation CLIST.
   b. On panel DSNTIPA1, select UPDATE mode.
   c. On panel DSNTIPT, change any data set names that are not correct.
   d. On panel DSNTIPB, select the ARCHIVE LOG DATA SET PARAMETERS option.
   e. On panel DSNTIPA, increase the value of RECORDING MAX.
   f. When the installation CLIST editing completes, rerun job DSNTIJUZ to recompile the system parameters.
4. Start DB2 with the -START DB2 command.

For more information on updating DB2 system parameters, see Part 2 of *DB2 Installation Guide*.

For instructions about adding an old archive data set, refer to “Changing the BSDS log inventory” on page 342. Also, see Part 3 of *DB2 Utility Guide and Reference* for additional information on the change log inventory utility.

## Unresolvable BSDS or log data set problem during restart

Use dual logging (active logs, archive logs, and bootstrap data sets) to reduce your efforts in resolving the problem described in this section.

***Problem:*** During restart of DB2, serious problems with the BSDS or log data sets were detected and cannot be resolved.

***Symptom:*** The following messages are issued:
DSNJ100I
DSNJ107I
DSNJ119I

Any of the following problems could be involved:

- A log data set is physically damaged.
- Both copies of a log data set could be physically damaged in the case of dual logging mode.
- A log data set could be lost.
- An archive log volume could have been reused even though it was still needed.
- A log data set could contain records that are not recognized by DB2 because they are logically broken.

***System action:*** DB2 cannot be restarted unless the following procedure is used:

***Operations management action:*** In serious cases such as this, it can be necessary to fall back to a prior shutdown level. If this procedure is used, all database changes between the shutdown point and the present will be lost, but all the data retained will be consistent within DB2.

If it is necessary to fall back, read “Preparing to recover to a prior point of consistency” on page 383.

If too much log information has been lost, use the alternative approach described in “Failure resulting from total or excessive loss of log data” on page 496.

## Preparing for recovery of restart

See “Preparing to recover to a prior point of consistency” on page 383 for preparation procedures.

## Performing the fall back to a prior shutdown point

1. When a failure occurs and you decide to fall back, use the print log map utility against the most current copy of the BSDS. Even if you are not able to do this, continue with the next step. (If you are unable to do this, an error message will be issued.)
2. Use access method services IMPORT to restore the backed-up versions of the BSDS and active log data sets.
3. Use the print log map utility against the copy of the BSDS with which DB2 is to be restarted.
4. Determine whether any archive log data sets must be deleted.
   - If you have a copy of the most current BSDS, compare it to the BSDS with which DB2 is to be restarted. Delete and uncatalog any archive log data sets that are listed in the most current BSDS but are not listed in the previous one. These archive log data sets are normal physical sequential (SAM) data sets. If you are able to do this step, continue with step 5.
   - If you were not able to print a copy of the most current BSDS and the archive logs are cataloged, use access method services LISTCAT to check for archive logs with a higher sequence number than the last archive log shown in the BSDS being used to restart DB2.
     - If no archive log data sets with a higher sequence number exist, you do not have to delete or uncatalog any data sets, and you can continue with step 5.
     - Delete and uncatalog all archive log data sets that have a higher sequence number than the last archive log data set in the BSDS being used to restart DB2. These archive log data sets are SAM data sets. Continue with the next step.

   If the archive logs are not cataloged, it is not necessary to uncatalog them.
5. Give the command START DB2. Use -START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped. If DDL is required, the creator might not be the same.
6. Now, determine what data needs to be recovered, what data needs to be dropped, what data can remain unchanged, and what data needs to be recovered to the prior shutdown point.
   - For table spaces and indexes that might have been changed after the shutdown point, use the DB2 RECOVER utility to recover these table spaces and indexes. They must be recovered in the order indicated in Part 2 of *DB2 Utility Guide and Reference*.
   - For data that has not been changed after the shutdown point (data used with RO access), it is not necessary to use RECOVER or DROP.
   - For table spaces that were deleted after the shutdown point, issue the DROP statement. These table spaces will not be recovered.

- Any objects created after the shutdown point should be re-created.

*All data that has potentially been modified after the shutdown point must be recovered.* If the RECOVER utility is not used to recover modified data, serious problems can occur because of data inconsistency.

If an attempt is made to access data that is inconsistent, any of the following events can occur (and the list is not comprehensive):

- It is possible to successfully access the correct data.
- Data can be accessed without DB2 recognizing any problem, but it might not be the data you want (the index might be pointing to the wrong data).
- DB2 might recognize that a page is logically incorrect and, as a result, abend the subsystem with an X'04E' abend completion code and an abend reason code of X'00C90102'.
- DB2 might notice that a page was updated after the shutdown point and, as a result, abend the requester with an X'04E' abend completion code and an abend reason code of X'00C200C1'.

7. Analyze the CICS log and the IMS log to determine the work that must be redone (work that was lost because of shutdown at the previous point). Inform all TSO users, QMF users, and batch users for which no transaction log tracking has been performed, about the decision to fall back to a previous point.
8. When DB2 is started after being shut down, indoubt units of recovery can exist. This occurs if transactions are indoubt when the command -STOP DB2 MODE (QUIESCE) is given. When DB2 is started again, these transactions will still be indoubt to DB2. IMS and CICS cannot know the disposition of these units of recovery.

   To resolve these indoubt units of recovery, use the command RECOVER INDOUBT.
9. If a table space was dropped and re-created after the shutdown point, it should be dropped and re-created again after DB2 is restarted. To do this, use SQL DROP and SQL CREATE statements.

   Do not use the RECOVER utility to accomplish this, because it will result in the old version (which can contain inconsistent data) being recovered.
10. If any table spaces and indexes were created after the shutdown point, these must be re-created after DB2 is restarted. There are two ways to accomplish this:
    - For data sets defined in DB2 storage groups, use the CREATE TABLESPACE statement and specify the appropriate storage group names. DB2 automatically deletes the old data set and redefines a new one.
    - For user-defined data sets, use access method services DELETE to delete the old data sets. After these data sets have been deleted, use access method services DEFINE to redefine them; then use the CREATE TABLESPACE statement.

## Failure resulting from total or excessive loss of log data

***Problem:*** Either all copies of the BSDS and logs have been destroyed or lost, or an excessive amount of the active log has been destroyed or lost.

***Symptom:*** Any messages or abends indicating that all or an excessive amount of log information has been lost.

***System action:*** None.

***Operations management action:*** Restart DB2 without any log data by following either the procedure in “Total loss of log” or “Excessive loss of data in the active log” on page 498.

## Total loss of log

Even if all copies of the BSDS and either the active or archive log or both have been destroyed or lost, DB2 can still be restarted, and data that belongs to that DB2 subsystem can still be accessed, provided that all system and user table spaces have remained intact and you have a recent copy of the BSDS. However, you must rely on your own sources to determine what data is inconsistent, because DB2 cannot provide any hints of inconsistencies. We assume that you still have other VSAM clusters on disk, such as the system databases DSNDB01, DSNDB04, and DSNB06, as well as user databases. For example, you might know that DB2 was dedicated to a few processes (such as utilities) during the DB2 session, and you might be able to identify the page sets they modified. If you cannot identify the page sets that are inconsistent, you must decide whether you are willing to assume the risk involved in restarting DB2 under those conditions. If you decide to restart, take the following steps:

1. Define and initialize the BSDSs. See step 2 in “Recovering the BSDS from a backup copy” on page 431.
2. Define the active log data sets using the access method services DEFINE function. Run utility DSNJLOGF to initialize the new active log data sets.
3. Prepare to restart DB2 using no log data. See “Deferring restart processing” on page 354.

   Each data and index page contains the log RBA of the last log record applied against the page. Safeguards within DB2 disallow a modification to a page that contains a log RBA that is higher than the current end of the log. There are two choices.

   a. Run the DSN1COPY utility specifying the RESET option to reset the log RBA in every data and index page. Depending on the amount of data in the subsystem, this process can take quite a long time. Because the BSDS has been redefined and reinitialized, logging begins at log RBA 0 when DB2 starts.

      If the BSDS is not reinitialized, logging can be forced to begin at log RBA 0 by constructing a conditional restart control record (CRCR) that specifies a STARTRBA and ENDRBA that are both equal to 0, as the following shows:

      ```
      CRESTART CREATE,STARTRBA=0,ENDRBA=0
      ```

      Continue with step 4.

   b. Determine the highest possible log RBA of the prior log. From previous console logs written when DB2 was operational, locate the last DSNJ001I message. When DB2 switches to a new active log data set, this message is written to the console, identifying the data set name and the highest potential log RBA that can be written for that data set. Assume that this is the value X'8BFFF'. Add one to this value (X'8C000'), and create a conditional restart control record specifying the change log inventory control statement as shown below:

      ```
      CRESTART CREATE,STARTRBA=8C000,ENDRBA=8C000
      ```

When DB2 starts, all phases of restart are bypassed and logging begins at log RBA X'8C000'. If this method is chosen, it is not necessary to use the DSN1COPY RESET option and a lot of time is saved.

4. Start DB2. Use -START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped.
5. After restart, resolve all inconsistent data as described in “Resolving inconsistencies resulting from conditional restart” on page 500.

## Excessive loss of data in the active log

By studying “Total loss of log” on page 497, a procedure can be developed for restarting that meets the requirements of the situation. Specifically, when an excessive amount of the active log has been lost, the procedure can be adapted to fit the situation, as described in “Total loss of log” on page 497. *Do not delete and redefine the BSDS*. Instead, proceed as follows:

1. Use the print log map utility (DSNJU004) against the copy of the BSDS with which DB2 is to be restarted.
2. Use the print log map output to obtain the data set names of all active log data sets. Use access method services LISTCAT to determine which active log data sets are no longer available or usable.
3. Use access method services DELETE to delete all active log data sets that are no longer usable.
4. Use access method services DEFINE to define new active log data sets. Run utility DSNJLOGF to initialize the new active log data sets. One active log data set must be defined for each one found to be no longer available or usable in step 2 above. Use the active log data set name found in the BSDS as the data set name for the access method services DEFINE statement.
5. Using the print log map utility (DSNJU004) output, note whether an archive log data set exists that contains the RBA range of the redefined active log data set. To do this, note the starting and ending RBA values for the active log data set that was newly redefined, and look for an archive log data set with the same starting and ending RBA values.

   If no such archive log data sets exist, then:

   a. Use the change log inventory utility (DSNJU003) DELETE statement to delete the newly redefined active log data sets from the BSDS active log data set inventory.
   b. Next, use the change log inventory utility (DSNJU003) NEWLOG statement to add the active log data set back into the BSDS active log data set inventory. Do not specify RBA ranges on the NEWLOG statement.

   If the corresponding archive log data sets exist, then there are two courses of action:

   - If you want to minimize the number of potential read operations against the archive log data sets, then use access method services REPRO to copy the data from each archive log data set into the corresponding active log data set. Make certain you copy the proper RBA range into the active log data set.

     Be sure that the active log data set is big enough to hold all the data from the archive log data set. When DB2 does an archive operation, it copies the log data from the active log data set to the archive log data set, then pads the archive log data set with binary zeroes to fill a block. In order for the access method services REPRO command to be able to copy all of the data from the archive log data set to a newly defined active log data set, the new active log data set might need to be bigger than the original one. For example, if the block size of the archive log data set is 28 KB, and the active log data set contains 80 KB of data, DB2 copies the 80 KB and pads the archive log data set with 4 KB of nulls to fill the last block. Thus, the archive log data set now contains 84 KB of data instead of 80 KB. In order for the access method services REPRO command to complete successfully, the active log data set must be able to hold 84 KB, rather than just 80 KB of data.

- If you are not concerned about read operations against the archive log data sets, then do the same two steps as indicated above (as though the archive data sets did not exist).

6. Choose the appropriate point for DB2 to start logging (X'8C000') as described in the preceding procedure.
7. To restart DB2 without using any log data, create a CRCR, as described for the change log inventory utility (DSNJU003) in Part 3 of *DB2 Utility Guide and Reference* .
8. Start DB2. Use -START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped.
9. After restart, resolve all inconsistent data as described in “Resolving inconsistencies resulting from conditional restart” on page 500.

This procedure will cause all phases of restart to be bypassed and logging to begin at log RBA X'8C000'. It will create a gap in the log between the highest RBA kept in the BSDS and X'8C000', and that portion of the log will be inaccessible.

No DB2 process can tolerate a gap, including RECOVER. Therefore, all data must be image copied after a cold start. Even data that is known to be consistent must be image copied again when a gap is created in the log.

There is another approach to doing a cold start that does not create a gap in the log. This is only a method for eliminating the gap in the physical record. It does not mean that you can use a cold start to resolve the logical inconsistencies. The procedure is as follows:

1. Locate the last valid log record by using DSN1LOGP to scan the log. (Message DSN1213I identifies the last valid log RBA.)
2. Begin at an RBA that is known to be valid. If message DSN1213I indicated that the last valid log RBA is at X'89158', round this value up to the next 4K boundary (X'8A000').
3. Create a CRCR similar to the following.

   ```
   CRESTART CREATE,STARTRBA=8A000,ENDRBA=8A000
   ```
4. Use -START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped.
5. Now, take image copies of all data for which data modifications were recorded beyond log RBA X'8A000'. If you do not know what data was modified, take image copies of all data.

   If image copies are not taken of data that has been modified beyond the log RBA used in the CRESTART statement, future RECOVER operations can fail or result in inconsistent data.

After restart, resolve all inconsistent data as described in Resolving inconsistencies resulting from conditional restart.

## Resolving inconsistencies resulting from conditional restart

When a conditional restart of the DB2 subsystem is done, the following can occur:

- Some amount of work is left incomplete.
- Some data is left inconsistent.
- Information about the status of the DB2 subsystem is made unusable.

## Inconsistencies in a distributed environment

In a distributed environment, when a DB2 system restarts, it indicates its restart status and the name of its recovery log to the systems with which it communicates. There are two possible conditions for restart status, *warm* and *cold*.

A cold status for restart means that the DB2 system has no memory of previous connections with its partner, and therefore has no memory of indoubt logical units of work. This includes all postponed aborted URs that end without resolution and any restart pending page sets and partitions are removed from restart pending state. The partner accepts the cold start connection and remembers the recovery log name of the cold starting DB2. If the partner has indoubt thread resolution requirements with the cold starting DB2, they cannot be achieved. The partner terminates its indoubt resolution responsibility with the cold starting DB2. However, as a participant, the partner has indoubt logical units of work that must be resolved manually. The DB2 system has an incomplete record of resolution responsibilities. It attempts to reconstruct as much resynchronization information as possible and displays the information in one or more DSNL438 or DSNL439 messages. The displayed information is then forgotten.

A warm status for restart means the DB2 system does have memory of previous connections with the partner and therefore does have memory of indoubt logical units of work. The exchange of recovery log names validates that the correct recovery logs are being used for indoubt resolution. Each partner indicates its recovery log name and the recovery log name it believes to be the one the other partner is using. A warm start connection where one system specifies a recovery log name that is different than the name remembered by the other system is rejected if indoubt resolution is required between the two partners.

## Procedures for resolving inconsistencies

The following section explains what must be done to resolve any inconsistencies that exist. Before reading this section, the procedures in the other sections of this chapter must be considered. Each one provides important steps that must be followed before using the information in this section.

The following three methods describe one or more steps that must be taken to resolve inconsistencies in the DB2 subsystem. Before using these methods, however, do the following:

1. Obtain image copies of all DB2 table spaces. You will need these image copies if any of the following conditions apply:
   - You did a cold start.
   - You did a conditional restart that altered or truncated the log.
   - The log is damaged.
   - Part of the log is no longer accessible.

   The first thing to do after a conditional restart is to take image copies of all DB2 table spaces, except those that are inconsistent. For those table spaces suspected of being inconsistent, resolve the inconsistencies and then obtain image copies of them.

A cold start might cause down-level page set errors. Some of these errors cause message DSNB232I to be displayed during DB2 restart. After you restart DB2, check the console log for down-level page set messages. If any of those messages exist, correct the errors before you take image copies of the affected data sets. Other down-level page set errors are not detected by DB2 during restart. If you use the COPY utility with the SHRLEVEL REFERENCE option to make image copies, the COPY utility will issue message DSNB232I when it encounters down-level page sets. Correct those errors and continue making image copies. If you use some other method to make image copies, you will find out about down-level errors during normal DB2 operation. "Recovery from down-level page sets" on page 435 describes methods for correcting down-level page set errors.

Pay particular attention to DB2 subsystem table spaces. If any are inconsistent, recover all of them in the order shown in the discussion on recovering catalog and directory objects in Part 2 of *DB2 Utility Guide and Reference*.

When a portion of the DB2 recovery log becomes inaccessible, all DB2 recovery processes have difficulty operating successfully, including restart, RECOVER, and deferred restart processing. Conditional restart allowed circumvention of the problem during the restart process. To ensure that RECOVER does not attempt to access the inaccessible portions of the log, secure a copy (either full or incremental) that does not require such access. A failure occurs any time a DB2 process (such as the RECOVER utility) attempts to access an inaccessible portion of the log. You cannot be sure which DB2 processes must use that portion of the recovery log, and, therefore, you must assume that all data recovery requires that portion of the log.

2. Resolve database inconsistencies. If you determine that the existing inconsistencies involve indexes only (not data), use the utility RECOVER INDEX. If the inconsistencies involve data (either user data or DB2 subsystem data), continue reading this section.

   Inconsistencies in DB2 subsystem databases DSNDB01 and DSNDB06 must be resolved before inconsistencies in other databases can be resolved. This is necessary because the subsystem databases describe all other databases, and access to other databases requires information from DSNDB01 and DSNDB06.

   If the table space that cannot be recovered (and is thus inconsistent) is being dropped, either all rows are being deleted or the table is not necessary. In either case, drop the table when DB2 is restarted, and do not bother to resolve the inconsistencies before restarting DB2.

Any one of the following three procedures can be used to resolve data inconsistencies. However, it is advisable to use one of the first two procedures because of the complexity of the third procedure.

## Method 1. Recover to a prior point of consistency

See "Recovering data to a prior point of consistency" on page 396 for a description of how to successfully prepare for and do data recovery to a prior point of consistency.

## Method 2. Re-create the table space

Take the following steps to drop the table space and reconstruct the data using the CREATE statement. This procedure is simple relative to "Method 3. Use the REPAIR utility on the data" on page 502. However, if you want to use this procedure, you need to plan ahead, because, when a table space is dropped, all tables in that table space, as well as related indexes, authorities, and views, are implicitly dropped. Be prepared to reestablish indexes, views, and authorizations, as well as the data content itself.

DB2 subsystem tables, such as the catalog and directory, cannot be dropped. Follow either "Method 1. Recover to a prior point of consistency" on page 501 or "Method 3. Use the REPAIR utility on the data" for these tables.

1. Issue an SQL DROP TABLESPACE statement for all table spaces suspected of being involved in the problem.
2. Re-create the table spaces, tables, indexes, synonyms, and views using SQL CREATE statements.
3. Grant access to these objects as it was granted prior to the time of the error.
4. Reconstruct the data in the tables.
5. Run the RUNSTATS utility on the data.
6. Use COPY to acquire a full image copy of all data.
7. Use the REBIND process on all plans that use the tables or views involved in this activity.

## Method 3. Use the REPAIR utility on the data

The third method for resolving data inconsistencies involves the use of the REPAIR utility. This method of resolving inconsistencies is not recommended unless the inconsistency is limited to a small number of data or index pages for the following reasons.

- For extensive data inconsistency, this method can be fairly time consuming and complex, making the procedure more error prone than the two methods described previously.
- DSN1LOGP can identify page sets that contain inconsistencies, but it cannot identify the specific data modifications involved in the inconsistencies within a given page set.
- DB2 provides no mechanism to tell users whether data is physically consistent or damaged. If the data is damaged physically, you might learn this when you attempt to use SQL to access the data and find that the data is inaccessible.

If you decide to use this method to resolve data inconsistencies, be sure to read the following section carefully, because it contains information that is important to the successful resolution of the inconsistencies.

***Considerations for using the REPAIR method:***

- Any pages that are on the logical page list (perhaps caused by this restart) cannot be accessed via the REPAIR utility. Because you have decided to use the REPAIR utility to resolve the inconsistency, give the command -START DATABASE (*dbase*) SPACENAM (*space*) ACCESS(FORCE), where *space* names the table space involved. That allows access to the data.
- As noted in "Recovering data to a prior point of consistency" on page 396, DB2 subsystem data (in the catalog and directory) exists in interrelated tables and table spaces. Data in DB2 subsystem databases cannot be modified via SQL, so the REPAIR utility must be used to resolve the inconsistencies that are identified.
- For a description of stored data and index formats, refer to Part 6 of *DB2 Diagnosis Guide and Reference*.
- DB2 stores data in data pages. The structure of data in a data page must conform to a set of rules for DB2 to be able to process the data accurately. Using a conditional restart process does not cause violations to this set of rules; but, if

violations existed prior to conditional restart, they will continue to exist after conditional restart. Therefore, use DSN1COPY with the CHECK option.

- DB2 uses several types of pointers in accessing data. Each type (indexes, hashes, and links) is described in Part 6 of *DB2 Diagnosis Guide and Reference*. Look for these pointers and manually ensure their consistency.

  Hash and link pointers exist in the DB2 directory database; link pointers also exist in the catalog database. DB2 uses these pointers to access data. During a conditional restart, it is possible for data pages to be modified without update of the corresponding pointers. When this occurs, one of the following things can happen:

  – If a pointer addresses data that is nonexistent or incorrect, DB2 abends the request. If SQL is used to access the data, a message identifying the condition and the page in question is issued.

  – If data exists but no pointer addresses it, that data is virtually invisible to all functions that attempt to access it via the damaged hash or link pointer. The data might, however, be visible and accessible by some functions, such as SQL functions that use some other pointer that was not damaged. As might be expected, this situation can result in inconsistencies.

  If a row containing a varying-length field is updated, it can increase in size. If the page in which the row is stored does not contain enough available space to store the additional data, the row is placed in another data page, and a pointer to the new data page is stored in the original data page. After a conditional restart, one of the following can occur.

  – The row of data exists, but the pointer to that row does not exist. In this case, the row is invisible and the data cannot be accessed.

  – The pointer to the row exists, but the row itself no longer exists. DB2 abends the requester when any operation (for instance, a SELECT) attempts to access the data. If termination occurs, one or more messages will be received that identify the condition and the page containing the pointer.

  When these inconsistencies are encountered, use the REPAIR utility to resolve them, as described in Part 2 of *DB2 Utility Guide and Reference*.

- If the log has been truncated, there can be problems changing data via the REPAIR utility. Each data and index page contains the log RBA of the last recovery log record that was applied against the page. DB2 does not allow modification of a page containing a log RBA that is higher than the current end of the log. If the log has been truncated and you choose to use the REPAIR utility rather than recovering to a prior point of consistency, the DSN1COPY RESET option must be used to reset the log RBA in every data and index page set to be corrected with this procedure.

- **This last step is imperative.** When all known inconsistencies have been resolved, full image copies of all modified table spaces must be taken, in order to use the RECOVER utility to recover from any future problems.

# Part 5. Performance monitoring and tuning

# Chapter 24. Planning your performance strategy

The first step toward improving performance is planning, which is the subject of this chapter.

For an overview of the whole book, see “Further topics in monitoring and tuning”.

For the elements of planning a strategy, see the following sections:
- “Managing performance in general” on page 518
- “Establishing performance objectives” on page 518
- “Planning for monitoring” on page 522
- “Reviewing performance data” on page 524
- “Tuning DB2” on page 526

Information on performance monitoring and tuning in a data sharing environment is presented in *DB2 Data Sharing: Planning and Administration*.

## Further topics in monitoring and tuning

“Chapter 25. Analyzing performance data” on page 527 is a general guide to analyzing and investigating performance issues.

“Chapter 26. Improving response time and throughput” on page 537 deals with space allocation, buffer pool and data set usage, processor resources, and response time reporting.

“Chapter 27. Tuning DB2 buffer, EDM, RID, and sort pools” on page 549 has recommendations for monitoring and tuning those objects.

“Chapter 28. Improving resource utilization” on page 579 deals with managing data sets, logging, disks, main storage, the resource limit facility, and performance options in MVS.

“Chapter 29. Managing DB2 threads” on page 619 deals with DB2 threads and workload management.

“Chapter 30. Improving concurrency” on page 643 deals with concurrency and locking.

“Chapter 31. Tuning your queries” on page 711 deals with writing queries that are as efficient as possible.

“Chapter 32. Maintaining statistics in the catalog” on page 765 deals with catalog statistics, reorganizing, and rebinding.

“Chapter 33. Using EXPLAIN to improve SQL performance” on page 789 deals with the principal means of monitoring access path selection and improving the performance of your SQL, especially queries.

“Chapter 34. Parallel operations and query performance” on page 841 deals specifically with monitoring and tuning parallel queries.

“Chapter 35. Tuning and monitoring in a distributed environment” on page 857 deals with performance in a distributed environment.

"Chapter 36. Monitoring and tuning stored procedures and user-defined functions" on page 873 deals with using stored procedures and user-defined functions efficiently.

Throughout this section, bear in mind the following:

- The emphasis is on performance objectives that can be reasonably measured by tools now available. That might not adequately serve your purpose. If, for example, you serve a diverse range of query users and want to measure "user satisfaction", you might need more than the techniques described here.
- DB2 is only a part of your overall system. Any change to programs, such as MVS, IMS, or CICS, that share your machine and I/O devices can affect how DB2 runs.
- The recommendations in this section are based on current knowledge of DB2 performance for "normal" circumstances and "typical" systems. Therefore, that this book provides the best or most appropriate advice for any specific site cannot be guaranteed. In particular, the advice in this section approaches situations from a performance viewpoint only; at some sites, other factors of higher priority may make some recommendations in this section inappropriate.
- The recommendations are general. Actual performance statistics are not included because such measurements are highly dependent on workload and system characteristics external to DB2.

## Managing performance in general

Managing the performance of any system involves the following steps:

1. Establish performance objectives.
2. Plan how to monitor performance.
3. Carry out the plan.
4. Analyze performance reports to decide whether the objectives have been met.
5. If performance is thoroughly satisfactory, use one of the following options:
   - Monitor less, because monitoring itself uses resources.
   - Continue monitoring to generate a history of performance to compare with future results.
6. If performance has not been satisfactory, take the following actions:
   a. Determine the major constraints in the system.
   b. Decide where you can afford to make trade-offs and which resources can bear an additional load. Nearly all tuning involves trade-offs among system resources.
   c. Tune your system by adjusting its characteristics to improve performance.
   d. Return to step 3 above and continue to monitor the system.

Periodically, or after significant changes to your system or workload, return to step 1, reexamine your objectives, and refine your monitoring and tuning strategy accordingly.

## Establishing performance objectives

How you define good performance for your DB2 subsystem depends on your particular data processing needs and their priority. Performance objectives should be realistic, in line with your budget, understandable, and measurable.

Common objectives include values for:

- Acceptable response time (a duration within which some percentage of all applications have completed)
- Average throughput (the total number of transactions or queries that complete within a given time)
- System availability, including mean time to failure and the durations of down times

Objectives such as those define the workload for the system and determine the requirements for resources—processor speed, amount of storage, additional software, and so on. Often, though, available resources limit the maximum acceptable workload, which requires revising the objectives.

***Service-level agreements:*** Presumably, your users have a say in your performance objectives. A mutual agreement on acceptable performance, between the data processing and user groups in an organization, is often formalized and called a *service-level agreement*. Service-level agreements can include expectations of query response time, the workload throughput per day, hour, or minute, and windows provided for batch jobs (including utilities). These agreements list criteria for determining whether or not the system is performing adequately.

For example, a service-level agreement might require that 90% of all response times sampled on a local network in the prime shift be under 2 seconds, or that the average response time not exceed 6 seconds even during peak periods. (For a network of remote terminals, consider substantially higher response times.)

Performance objectives must reflect not only elapsed time, but also the amount of processing expected. Consider whether to define your criteria in terms of the average, the ninetieth percentile, or even the worst-case response time. Your choice can depend on your site’s audit controls and the nature of the workloads.

## Defining the workload

To define the workload of the system, begin by determining the type of workload. For each type, describe a preliminary workload profile that includes:

- A definition of the workload type in terms of its function and its volume. You are likely to have many workloads that perform the same general function (for example, order entry) and have an identifiable workload profile. Other workload types could be SPUFI or QMF queries. For the volume of a workload that is already processed by DB2, use the summary of its volumes from the DB2 statistics trace.
- The relative priority of the type, including periods during which the priorities change.
- The resources required to do the work, including physical resources managed by the operating system (such as real storage, disk I/O, and terminal I/O) and logical resources managed by the subsystem (such as control blocks and buffers).

Before installing DB2, gather design data during the phases of initial planning, external design, internal design, and coding and testing. Keep reevaluating your performance objectives with that information.

## Initial planning

Begin establishing resource requirements by estimating the quantities listed below, however uncertain they might seem at this stage.

***For transactions:***

- Availability transaction managers, such as IMS or CICS
- Number of message pairs (inputs and outputs to a terminal) for each user function
- Line speeds (bits per second) for remote terminals
- Number of terminals and operators needed to achieve the required throughput
- Maximum rate of workloads per minute, hour, day, or week
- Number of I/O operations per user workload (disks and terminals)
- Average and maximum processor usage per workload type and total work load
- Size of tables
- Effects of objectives on operations and system programming

***For query use:***
- Time required to key in user data
- Online query processing load
- Limits to be set for the query environment or preformatted queries
- Size of tables
- Effects of objectives on operations and system programming

***For batch processing:***
- Batch windows for data reorganization, utilities, data definition activities, and BIND processing
- Batch processing load
- Length of batch window
- Number of records to process, data reorganization activity, use of utilities, and data definition activity
- Size of tables
- Effects of objectives on operations and system programming

Look at the base estimate to find ways of reducing the workload. Changes in design at this stage, before contention with other programs, are likely to be the most effective. Later, you can compare the actual production profile against the base.

## Translating resource requirements into objectives

For each workload type, convert the estimates of resource requirements into measurable objectives. Include statements about the throughput rates to be supported (including any peak periods) and the internal response time profiles to be achieved. Make assumptions about I/O rates, paging rates, and workloads. Include the following factors:

- System response time. You cannot guarantee requested response times before any of the design has been done. Hence, plan to review your performance targets along with designing and implementing the system.

  Response times can vary for many reasons. Therefore, include acceptable tolerances in your descriptions of targets. Remember that distributed data processing adds overhead at both the local and remote locations.

  Exclude from the targets any unusual applications that have exceptionally heavy requirements for processing or database access, or establish individual targets for those applications.
- Network response time. Responses in the processor are likely to be in fractions of seconds, while responses in the network can amount to seconds. This means that a system can never deliver fast responses through an overloaded network, however fast the processor. Queries over the network to remote systems slow response further.
- Disk response time. I/O operations are generally responsible for much internal processing time. Consider all I/O operations that affect a workload.

- Existing workload. Consider the effects of additional work on existing applications. In planning the capacity of the system, consider the total load on each major resource, not just the load for the new application.
- Business factors. When calculating performance estimates, concentrate on the expected peak throughput rate. Allow for daily peaks (for example, after receipt of mail), weekly peaks (for example, a Monday peak after weekend mail), and seasonal peaks as appropriate to the business. Also allow for peaks of work after planned interruptions, such as preventive maintenance periods and public holidays. Remember that the availability of input data is one of the constraints on throughput.

### External design

During the external design phase, you must:

1. Estimate the network, processor, and disk subsystem workload.
2. Refine your estimates of logical disk accesses. Ignore physical accesses at this stage; one of the major difficulties will be determining the number of I/Os per statement.

### Internal design

During the internal design phase, you must:

1. Refine your estimated workload against the actual workload.
2. Refine disk access estimates against database design. After internal design, you can define physical data accesses for application-oriented processes and estimate buffer hit ratios.
3. Add the accesses for DB2 work file database, DB2 log, program library, and DB2 sorts.
4. Consider whether additional processor loads will cause a significant constraint.
5. Refine estimates of processor usage.
6. Estimate the internal response time as the sum of processor time and synchronous I/O time or as asynchronous I/O time, whichever is larger.
7. Prototype your DB2 system. Before committing resources to writing code, you can create a small database, update the statistics stored in the DB2 catalog tables, run SELECT, UPDATE, INSERT, DELETE, and EXPLAIN statements, and examine the results. This method, which relies on production-level statistics, allows you to prototype index design and evaluate access path selection for an SQL statement. Buffer pool size, the presence or absence of the DB2 sort facility, and, to a lesser extent, processor size are also factors that impact DB2 processing.
8. Use DB2 estimation formulas to develop estimates for processor resource consumption and I/O costs for application processes that are high volume or complex.

### Coding and testing

During this phase:

1. Refine the internal design estimates of disk and processing resources.
2. Run the monitoring tools you have selected and check the results against the estimates. You might use a terminal network simulator such as TeleProcessing Network Simulator (TPNS) to test the system and simulate load conditions.

## Post-development review

When you are ready to test the complete system, review its performance in detail. Take the following steps to complete your performance review:

1. Validate system performance and response times against the objectives.

2. Identify resources whose usage requires regular monitoring.
3. Incorporate the observed figures into future estimates. This step requires:
   a. Identifying discrepancies from the estimated resource usage
   b. Identifying the cause of the discrepancies
   c. Assigning priorities to remedial actions
   d. Identifying resources that are consistently heavily used
   e. Setting up utilities to provide graphic representation of those resources
   f. Projecting the processor usage against the planned future system growth to ensure that adequate capacity will be available
   g. Updating the design document with the observed performance figures
   h. Modifying the estimation procedures for future systems

You need feedback from users and might have to solicit it. Establish reporting procedures and teach your users how to use them. Consider logging incidents such as these:

- System, line and transaction or query failures
- System unavailable time
- Response times that are outside the specified limits
- Incidents that imply performance constraints, such as deadlocks, deadlock abends, and insufficient storage
- Situations, such as recoveries, that use additional system resources

The data logged should include the time, date, location, duration, cause (if it can be determined), and the action taken to resolve the problem.

## Planning for monitoring

Your plan for monitoring DB2 should include:

- A master schedule of monitoring. Large batch jobs or utility runs can cause activity peaks. Coordinate monitoring with other operations so that it need not conflict with unusual peaks, unless that is what you want to monitor.
- The kinds of analysis to be performed and the tools to be used. Document the data that is extracted from the monitoring output.

  Some of the reports discussed later in this chapter are derived from the products described in “Appendix F. Using tools to monitor performance” on page 1029. These reports can be produced using Performance Reporter for MVS (formerly known as EPDM), DB2 Performance Monitor (DB2 PM), other reporting tools, manual reduction, or a program of your own that extracts information from standard reports.
- A list of people who should review the results. The results of monitoring and the conclusions based on them should be available to the user support group and to system performance specialists.
- A strategy for tuning DB2. Describe how often changes are permitted and standards for testing their effects. Include the tuning strategy in regular system management procedures.

  Tuning recommendations could include generic database and application design changes. You should update development standards and guidelines to reflect your experience and to avoid repeating mistakes.

Typically, your plan provides for four levels of monitoring: continuous, periodic, detailed, and exception. These levels are discussed in the sections that follow. “A monitoring strategy” on page 524 describes a plan that includes all of these levels.

## Continuous monitoring

For monitoring the basic load of the system, try continually running classes 1, 3, and 4 of the DB2 statistics trace and classes 1 and 3 of the DB2 accounting trace. In the data you collect, look for statistics or counts that differ from past records. Pay special attention to peak periods of activity, both of any new application and of the system as a whole.

Running accounting class 2 as well as class 1 allows you to separate DB2 times from application times.

With CICS, there is less need to run with accounting class 2. Application and non-DB2 processing take place under the CICS main TCB. Because SQL activity takes place under the SQL TCB, the class 1 and class 2 times are generally close. The CICS attachment work is spread across class 1, class 2, and not-in-DB2 time. Class 1 time thus reports on the SQL TCB time and some of the CICS attachment. If you are concerned about class 2 overhead and you use CICS, you can generally run without turning on accounting class 2.

## Periodic monitoring

A typical periodic monitoring interval of about ten minutes provides information on the workload achieved, resources used, and significant changes to the system. In effect, you are taking “snapshots” at peak loads and under normal conditions. It is always useful to monitor peak periods when constraints and response-time problems are more pronounced.

The current peak is also a good indicator of the future average. You might have to monitor more frequently at first to confirm that expected peaks correspond with actual ones. Do not base conclusions on one or two monitoring periods, but on data from several days representing different periods.

Both continuous and periodic monitoring serve to check system throughput, utilized resources (processor, I/Os, and storage), changes to the system, and significant exceptions that might affect system performance. You might notice that subsystem response is becoming increasingly sluggish, or that more applications fail from lack of resources (such as from locking contention or concurrency limits). You also might notice an increase in the processor time DB2 is using, even though subsystem responses seem normal. In any case, if the subsystem continues to perform acceptably and you are not having any problems, DB2 might not need further tuning.

For periodic monitoring, gather information from MVS, the transaction manager, and DB2 itself. To compare the different results from each source, monitor each for the same period of time. Because the monitoring tools require resources, you need to consider processor overhead for using these tools. See “Minimize the use of DB2 traces” on page 545 for information on DB2 trace overhead.

## Detailed monitoring

Add detailed monitoring to periodic monitoring when you discover or suspect a problem. Use it also to investigate areas not covered periodically.

If you have a performance problem, first verify that it is not caused by faulty design of an application or database. If you suspect a problem in application design, consult Part 4 of *DB2 Application Programming and SQL Guide*; for information about database design, see “Part 2. Designing a database: advanced topics” on page 27.

If you believe that the problem is caused by the choice of system parameters, I/O device assignments, or other factors, begin monitoring DB2 to collect data about its internal activity. “Appendix F. Using tools to monitor performance” on page 1029 suggests various techniques and methods.

If you have access path problems, refer to “Chapter 33. Using EXPLAIN to improve SQL performance” on page 789 for information.

### Exception monitoring

Exception monitoring looks for specific exceptional values or events, such as very high response times or deadlocks. Perform exception monitoring for response-time and concurrency problems. For an example, see “Analyzing a concurrency scenario” on page 702.

### A monitoring strategy

Consider the following cost factors when planning for monitoring and tuning:

- Trace overhead
- Trace data reduction and reporting times
- Time spent on report analysis and tuning action

“Minimize the use of DB2 traces” on page 545 discusses overhead for global, accounting, statistics, audit, and performance traces.

## Reviewing performance data

Inspect your performance data to determine whether performance has been satisfactory, to identify problems, and to evaluate the monitoring process. When establishing requirements and planning to monitor performance, also plan how to review the results of monitoring.

Plan to review the performance data systematically. Review daily data weekly and weekly data monthly; review data more often if a report raises questions that require checking. Depending on your system, the weekly review might require about an hour, particularly after you have had some experience with the process and are able to locate quickly any items that require special attention. The monthly review might take half a day at first, less time later on. But when new applications are installed, workload volumes increased, or terminals added, allow more time for review.

Review the data on a gross level, looking for problem areas. Review details only if a problem arises or if you need to verify measurements.

When reviewing performance data, try to identify the basic pattern in the workload, and then identify variations of the pattern. After a certain period, discard most of the data you have collected, but keep a representative sample. For example, save the report from the last week of a month for three months; at the end of the year, discard all but the last week of each quarter. Similarly, keep a representative selection of daily and monthly figures. Because of the potential volume of data, consider using EPDM or a similar tool to track historical data in a manageable form.

## Typical review questions

Use the questions listed below as a basis for your own checklist. They are not limited strictly to performance items, but your historical data can provide most of their answers. Pointers to more information are also listed.

**How often was each function used?**

1. Considering variations in the workload mix over time, are the monitoring times appropriate?
2. Should monitoring be done more frequently during the day, week, or month to verify this?

See "Accounting trace" on page 1034.

**How were processor and I/O resources used?**

1. Has usage increased for functions that run at a higher priority than DB2 tasks? Examine CICS, IMS, MVS, JES, VTAM (if running above DB2), and overall I/O because of the lower-priority regions. Evaluate the effectiveness of I/O scheduling priority decisions as appropriate. See also "I/O scheduling priority" on page 615 for more information on I/O priority scheduling.
2. Is the report of processor usage consistent with previous observations?
3. Are scheduled batch jobs able to run successfully?
4. Do any incident reports show that the first invocation of a function takes much longer than later ones? This increased time can happen when programs have to open data sets.

See "Monitoring system resources" on page 1031, "Using MVS, CICS, and IMS tools" on page 1030, and "Statistics trace" on page 1034.

**To what degree was disk used?**

Is the number of I/O requests increasing? DB2 records both physical and logical requests. The number of physical I/Os depend on the configuration of indexes, the data records per control interval, and the buffer allocations.

See "Monitoring system resources" on page 1031 and "Statistics trace" on page 1034.

**How much real storage was used?**

Is the paging rate increasing? Adequate real storage is very important for DB2 performance.

See "Monitoring system resources" on page 1031.

**To what extent were DB2 log resources used?**

1. Is the log subject to undue contention from other data sets? In particular, is the log on the same drive as any resource whose updates are logged?

   It is bad practice to put a recoverable (updated) resource and a log on the same drive—if that drive fails, you lose both the resource and the log, and you are unable to carry out forward recovery.
2. What's the I/O rate for requests and physical blocks on the log?

See "Statistics trace" on page 1034.

**Do any figures indicate design, coding, or operational errors?**

1. Are disk, I/O, log, or processor resources heavily used? If so, was that heavy use expected at design time? If not, can the heavy use be explained in terms of heavier use of workloads?

2. Is the heavy usage associated with a particular application? If so, is there evidence of planned growth or peak periods?
3. What are your needs for concurrent read/write and query activity?
4. How often do locking contentions occur?
5. Are there any disk, channel, or path problems?
6. Are there any abends or dumps?

See “Monitoring system resources” on page 1031, “Statistics trace” on page 1034, and “Accounting trace” on page 1034.

**Were there any bottlenecks?**

1. Were any critical thresholds reached?
2. Are any resources approaching high utilization?

See “Monitoring system resources” on page 1031 and “Accounting trace” on page 1034.

### Are your performance objectives reasonable?

After beginning to monitor, you need to find out if the objectives themselves are reasonable. Are they achievable, given the hardware available? Are they based upon actual measurements of the workload?

When you measure performance against initial objectives and report the results to users, identify any systematic differences between the measured data and what the user sees. This means investigating the differences between internal response time (seen by DB2) and external response time (seen by the end user). If the measurements differ greatly from the estimates, revise response-time objectives for the application, upgrade your system, or plan a reduced application workload. If the difference is not too large, however, you can begin tuning the entire system.

## Tuning DB2

Tuning DB2 can involve reassigning data sets to different I/O devices, spreading data across a greater number of I/O devices, running the RUNSTATS utility and rebinding applications, creating indexes, or modifying some of your subsystem parameters. For instructions on modifying subsystem parameters, see Part 2 of *DB2 Installation Guide*.

Tuning your system usually involves making trade-offs between DB2 and overall system resources.

After modifying the configuration, monitor DB2 for changes in performance. The changes might correct your performance problem. If not, repeat the process to determine whether the same or different problems exist.

# Chapter 25. Analyzing performance data

This chapter includes the following topics:

1. An overview of problem investigation and analysis, in "Investigating the problem overall"
2. A description of a major tool for analyzing problems in DB2, in "Reading accounting reports from DB2 PM" on page 528
3. A suggested procedure for analyzing problems within DB2, in "A general approach to problem analysis in DB2" on page 533

## Investigating the problem overall

When analyzing performance data, keep in mind that almost all symptoms of poor performance are magnified when contention occurs. For example, if there is a slowdown in disk operations:

- Transactions can pile up, waiting for data set activity.
- Transactions can wait for I/O and locks.
- Paging can be delayed.

In addition, more transactions in the system means greater processor overhead, greater virtual-storage demand, and greater real-storage demand.

In such situations, the system shows heavy use of *all* its resources. However, it is actually experiencing typical system stress, with a constraint that is yet to be found.

## Looking at the entire system

Start by looking at the overall system before you decide that you have a problem in DB2. In general, look in some detail to see why application processes are progressing slowly, or why a given resource is being heavily used. The best tool for that is the resource measurement facility (RMF™) of MVS.

## Beginning to look at DB2

Within DB2, the performance problem is either poor response time or an unexpected and unexplained high use of resources. Check factors such as total processor usage, disk activity, and paging.

First, get a picture of task activity, from classes 1 and 3 of the accounting trace, and then focus on particular activities, such as specific application processes or a specific time interval. You might see problems such as these:

- Slow response time. You could look at detailed traces of one slow task, a problem for which there could be several reasons. For instance, the users could be trying to do too much work with certain applications, work that clearly takes time, and the system simply cannot do all the work that they want done.
- Real storage constraints. Applications progress more slowly than expected because of paging interrupts. The constraints show as delays between successive requests recorded in the DB2 trace.
- Contention for a particular function. For example, there might be a wait on a particular data set, or a certain application might cause many application processes to put the same item in their queues. Use the DB2 performance trace to distinguish most of these cases.

To determine whether the problem is inside or outside DB2, activate classes 2 and 3 of the accounting trace for the troublesome application. For information about packages or DBRMs, run accounting trace classes 7 and 8. Compare the elapsed times for accounting classes 1 and 2.

A number greater than 1 in the QXMAXDEG field of the accounting trace indicates that parallelism was used. There are special considerations for interpreting such records, as described in “Monitoring parallel operations” on page 850.

The easiest way to read and interpret the trace data is through the reports produced by DB2 Performance Monitor (DB2 PM). If you do not have DB2 PM or an equivalent program, refer to “Appendix D. Interpreting DB2 trace output” on page 981 for information about the format of data from DB2 traces.

You can also use the tools for performance measurement described in “Appendix F. Using tools to monitor performance” on page 1029 to diagnose system problems. See that appendix also for information on analyzing the DB2 catalog and directory.

## Reading accounting reports from DB2 PM

You can obtain DB2 PM reports of accounting data in long or short format and in various levels of detail. The examples in this book are based on the default layouts, which might have been modified for your installation. Furthermore, the DB2 PM reports have been reformatted or modified for this publication. Refer to *DB2 PM for OS/390 Report Reference Volume 1* and *DB2 PM for OS/390 Report Reference Volume 2* for an exact description of each report. See “Accounting for nested activities” on page 879 for information on time results for triggers, stored procedures, and user-defined functions.

## The accounting report—short

***General capabilities:*** The DB2 PM accounting report, short layout, allows you to monitor application distribution, resources used by each major group of applications, and the average DB2 elapsed time for each major group. The report summarizes application-related performance data and orders the data by selected DB2 identifiers.

Monitoring application distribution helps you to identify the most frequently used transactions or queries, and is intended to cover the 20% of the transactions or queries that represent about 80% of the total work load. The TOP list function of DB2 PM lets you identify the report entries that represent the largest user of a given resource.

To get an overall picture of the system work load, you can use the DB2 PM GROUP command to group several DB2 plans together.

You can use the accounting report, short layout, to:

- Monitor the effect of each application or group on the total work load
- Monitor, in each application or group:
  – DB2 response time (elapsed time)
  – Resources used (processor, I/Os)
  – Lock suspensions
  – Application changes (SQL used)
  – Usage of packages and DBRMs
  – Processor, I/O wait, and lock wait time for each package

An accounting report in the short format can list results in order by package. Thus you can summarize package or DBRM activity independently of the plan under which the package or DBRM executed.

Only class 1 of the accounting trace is needed for a report of information only by plan. Classes 2 and 3 are recommended for additional information. Classes 7 and 8 are needed to give information by package or DBRM.

## The accounting report—long

Use the DB2 PM accounting report, short layout, to monitor your applications. Use the DB2 PM accounting report, long layout, when an application seems to have a problem, and you need a more detailed analysis. For a partial example of an accounting report, long layout, see Figure 56.

```
PLANNAME: PU22301
 AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS    AVERAGE TIME  AV.EVENT    HIGHLIGHTS
 ------------  ----------  ----------  ----------    --------------------   ------------  --------    -------------------------
 ELAPSED TIME    5.773449    3.619543         N/P    LOCK/LATCH(DB2+IRLM)  A    1.500181      1.09    #OCCURRENCES    :     80
  NON-NESTED     2.014711    1.533210         N/A    SYNCHRON. I/O  B           0.002096      0.13    #ALLIEDS        :     80
  STORED PROC    3.758738    2.086333         N/A     DATABASE I/O              0.000810      0.09    #ALLIEDS DISTRIB:     80
  UDF            0.000000    0.000000         N/A     LOG WRITE I/O             0.001286      0.04    #DBATS          :     80
  TRIGGER        0.000000    0.000000         N/A    OTHER READ I/O  D          0.000000      0.00    #DBATS DISTRIB. :      0
                                                     OTHER WRTE I/O  E          0.000000      0.00    #NO PROGRAM DATA:      0
 CPU TIME        0.141721 C  0.093469         N/P    SER.TASK SWTCH  F          0.860814      1.04    #NORMAL TERMINAT:     80
  AGENT          0.141721    0.093469         N/P     UPDATE COMMIT             0.010989      0.06    #ABNORMAL TERMIN:      0
   NON-NESTED    0.048918    0.004176         N/A     OPEN/CLOSE                0.448021      0.20    #CP/X PARALLEL  :      0
   STORED PROC   0.092802    0.089294         N/A     SYSLGRNG REC              0.193708      0.61    #IO PARALLELISM :      0
   UDF           0.000000    0.000000         N/A     EXT/DEL/DEF               0.160772      0.01    #INCREMENT. BIND:      0
   TRIGGER       0.000000    0.000000         N/A     OTHER  SERVICE            0.047324      0.16    #COMMITS        :     80
  PAR.TASKS      0.000000    0.000000         N/A    ARC.LOG(QUIES)  G          0.000000      0.00    #ROLLBACKS      :      0
                                                     ARC.LOG READ  H            0.000000      0.00    MAX SQL CASC LVL:      1
 SUSPEND TIME         N/A    2.832920         N/A    STORED PROC                0.129187      0.04     UPDATE/COMMIT  :   8.66
  AGENT               N/A    2.832920         N/A    UDF SCHEDULE               0.000000      0.00     SYNCH I/O AVG:0.0161123
  PAR.TASKS           N/A    0.000000         N/A    DRAIN LOCK  I              0.000000      0.00
                                                     CLAIM RELEASE  J           0.000000      0.00
 NOT ACCOUNT. L       N/A    0.693154         N/A    PAGE LATCH  K              0.000000      0.00
 DB2 ENT/EXIT         N/A        8.96         N/A    NOTIFY MSGS.               0.000000      0.00
 EN/EX-STPROC         N/A       41.74         N/A    GLOBAL CONT.               0.340642      7.37
 EN/EX-UDF            N/A         N/A         N/P    FORCE-AT-COMMIT            0.000000      9.67
 DCAPY.DESCR.         N/A         N/A         N/P    ASYNCH IXL REQUESTS        0.000000
 LOG EXTRACT.         N/A         N/A         N/P    TOTAL CLASS 3              2.832920
SQL DML    AVERAGE     TOTAL    SQL DCL            TOTAL    SQL DDL     CREATE    DROP   ALTER    LOCKING          AVERAGE     TOTAL
--------  --------  --------    --------------  --------    ----------  ------  ------  ------    --------------  --------  --------
SELECT        1.00        80    LOCK TABLE             0    TABLE            0       0       0    TIMEOUTS            0.00         0
INSERT        1.00        80    GRANT                  0    TEMP TABLE       0     N/A     N/A    DEADLOCKS           0.00         0
UPDATE        6.66       533    REVOKE                 0    AUX TABLE        0     N/A     N/A    ESCAL.(SHARED)      0.00         0
DELETE        1.00        80    SET CURR.SQLID         0    INDEX            0       0       0    ESCAL.(EXCLUS)      0.00         0
                                SET HOST VAR.          0    TABLESPACE       0       0       0    MAX LOCKSHELD       8.47        15
DESCRIBE      0.00         0    SET CUR.DEGREE         0    DATABASE         0       0       0    LOCK REQUEST       31.74      2539
DESC.TBL      0.00         0    SET RULES              0    STOGROUP         0       0       0    UNLOCK REQUEST      2.13       170
PREPARE       0.00         0    SET CURR.PATH          0    SYNONYM          0       0     N/A    QUERY REQUEST       0.00         0
OPEN          2.00       160    CONNECT TYPE 1         0    VIEW             0       0     N/A    CHANGE REQUEST     10.46       837
FETCH         8.66       693    CONNECT TYPE 2         0    ALIAS            0       0     N/A    OTHER REQUEST       0.00         0
CLOSE         0.00         0    SET CONNECTION         0    PACKAGE        N/A       0     N/A    LOCK SUSPENS.       0.31        25
                                RELEASE                0    PROCEDURE        0       0       0    LATCH SUSPENS.      0.15        12
                                CALL                   0    FUNCTION         0       0       0    OTHER SUSPENS.      0.15        12
DML-ALL      20.32      1626    ASSOC LOCATORS         0    TRIGGER          0       0     N/A    TOTAL SUSPENS.      0.46        37
                                ALLOC CURSOR           0    DIST TYPE        0       O     N/A
                                HOLD LOCATOR           0
                                FREE LOCATOR           0    TOTAL            0       0     N/A
                                DCL-ALL                0
                                                            RENAME TBL       0
```

⋮

*Figure 56. Partial accounting report, long layout*

### Major items on the report

In analyzing a detailed accounting report, consider the following components of response time. (Fields of the report that are referred to are labeled in Figure 56.)

***Class 1 elapsed time:*** Compare this with the CICS or IMS transit times:

- In CICS, you can use CMF to find the attach and detach times; use this time as the transit time.
- In IMS, use the PROGRAM EXECUTION time reported in IMS Performance Analyzer.

Differences between these CICS or IMS times, and the DB2 accounting times arise mainly because the DB2 times do not include:

- Time before the first SQL statement
- DB2 create thread
- DB2 terminate thread

Differences can also arise from thread reuse in CICS or IMS, or through multiple commits in CICS. If the class 1 elapsed time is significantly less than the CICS or IMS time, check the report from EPDM, IMS Performance Analyzer, or equivalent reporting tool to find out why. Elapsed time can occur:

- In DB2, during sign-on, create, or terminate thread
- Outside DB2, during CICS or IMS processing

For CICS, the transaction could have been waiting outside DB2 for a thread. Issue the DSNC DISPLAY STAT command to investigate this possibility. The column **W/P**, which is displayed as part of the output from DSNC DISPLAY STAT, contains the number of times all available threads for the RCT entry were busy and the transaction had to wait (THREADWAIT=YES or TWAIT=YES) or was diverted to the pool (THREADWAIT(POOL) or TWAIT=POOL).

***Not-in-DB2 time:*** This is time calculated as the difference between the class 1 and the class 2 elapsed time. It is time spent outside DB2, but within the DB2 accounting interval. A lengthy time can be caused by thread reuse, which can increase class 1 elapsed time, or a problem in the application program, CICS, IMS, or the overall system.

***Lock/latch suspension time:*** This shows contention for DB2 resources. If contention is high, check the locking summary section of the report, and then proceed with the locking reports. For more information, see “Analyzing a concurrency scenario” on page 702.

In the DB2 PM accounting report, see the field LOCK/LATCH(DB2+IRLM) ( **A** ).

***Synchronous I/O suspension time:*** This is the total application wait time for synchronous I/Os. It is the total of Database I/O and Log Write I/O. In the DB2 PM accounting report, check the number reported for SYNCHRON. I/O ( **B** ).

If the number of synchronous read or write I/Os is higher than expected, check for:

- A change in the access path to data. If you have data from accounting trace class 8, the number of synchronous and asynchronous read I/Os is available for individual packages. Determine which package or packages have unacceptable counts for synchronous and asynchronous read I/Os. Activate the necessary performance trace classes for the DB2 PM SQL activity reports to identify the SQL statement or cursor that is causing the problem. If you suspect that your application has an access path problem, see “Chapter 33. Using EXPLAIN to improve SQL performance” on page 789.
- Changes in the application. Check the “SQL ACTIVITY” section and compare with previous data. There might have been some inserts that changed the

amount of data. Also, check the names of the packages or DBRMs being executed to determine if the pattern of programs being executed has changed.
- Pages might be out of order so that sequential detection is not used, or data might have been moved to other pages. Run the REORG utility in these situations.
- A system-wide problem in the database buffer pool. Refer to “Using DB2 PM to monitor buffer pool statistics” on page 567.
- A RID pool failure. Refer to “Increasing RID pool size” on page 574.
- A system-wide problem in the EDM pool. Refer to “Tuning the EDM pool” on page 570.

If I/O time is greater than expected, and not caused by more read I/Os, check for:
- Synchronous write I/Os. See “Using DB2 PM to monitor buffer pool statistics” on page 567.
- I/O contention. In general, each synchronous read I/O typically takes from 10 to 25 milliseconds, depending on the disk device. This estimate assumes that there are no prefetch or deferred write I/Os on the same device as the synchronous I/Os. Refer to “Monitoring I/O activity of data sets” on page 598.

***Processor resource consumption:*** The problem might be caused by DB2 or IRLM traces, or by a change in access paths. In the DB2 PM accounting report, DB2 processor resource consumption is indicated in the field for class 2 CPU TIME ( **C** ).

***Other read suspensions:*** The accumulated wait time for read I/O done under a thread other than this one. It includes time for:
- Sequential prefetch
- List prefetch
- Sequential detection
- Synchronous read I/O performed by a thread other than the one being reported

As a rule of thumb, an asynchronous read I/O for sequential prefetch or sequential detection takes 0.4 to 2 milliseconds per page. For list prefetch, the rule of thumb is 1 to 4 milliseconds per page.

In the DB2 PM accounting report, other read suspensions are reported in the field OTHER READ I/O ( **D** ).

***Other write suspensions:*** The accumulated wait time for write I/O done under a thread other than this one. It includes time for:
- Asynchronous write I/O
- Synchronous write I/O performed by a thread other than the one being reported

As a rule of thumb, an asynchronous write I/O takes 1 to 4 milliseconds per page.

In the DB2 PM accounting report, other read suspensions are reported in the field OTHER WRTE I/O ( **E** ).

***Service task suspensions:*** The accumulated wait time from switching synchronous execution units, by which DB2 switches from one execution unit to another. The most common contributors to service task suspensions are:
- Wait for commit processing for updates (UPDATE COMMIT)
- Wait for OPEN/CLOSE service task (including HSM recall)
- Wait for SYSLGRNG recording service task

- Wait for data set extend/delete/define service task (EXT/DEL/DEF)
- Wait for other service tasks (OTHER SERVICE)

In the DB2 PM accounting report, the total of this information is reported in the field SER.TASK SWTCH ( F ). The field is the total of the five fields that follow it. If several types of suspensions overlap, the sum of their wait times can exceed the total clock time that DB2 spends waiting. Therefore, when service task suspensions overlap other types, the wait time for the other types of suspensions is not counted.

***Archive log mode (QUIESCE):*** The accumulated time the thread was suspended while processing ARCHIVE LOG MODE(QUIESCE). In the DB2 PM accounting report, this information is reported in the field ARCH.LOG (QUIES) ( G ).

***Archive log read suspension:*** This is the accumulated wait time the thread was suspended while waiting for a read from an archive log on tape. In the DB2 PM accounting report, this information is reported in the field ARCHIVE LOG READ ( H ).

***Drain lock suspension:*** The accumulated wait time the thread was suspended while waiting for a drain lock. If this value is high, see “Installation options for wait times” on page 665, and consider running the DB2 PM locking reports for additional detail. In the DB2 PM accounting report, this information is reported in the field DRAIN LOCK ( I ).

***Claim release suspension:*** The accumulated wait time the drainer was suspended while waiting for all claim holders to release the object. If this value is high, see “Installation options for wait times” on page 665, and consider running the DB2 PM locking reports for additional details.

In the DB2 PM accounting report, this information is reported in the field CLAIM RELEASE ( J ).

***Page latch suspension:*** This field shows the accumulated wait time because of page latch contention. As an example, when the RUNSTATS and COPY utilities are run with the SHRLEVEL(CHANGE) option, they use a page latch to serialize the collection of statistics or the copying of a page. The page latch is a short duration “lock”. If this value is high, the DB2 PM locking reports can provide additional data to help you determine which object is the source of the contention.

In the DB2 PM accounting report, this information is reported in the field PAGE LATCH ( K ).

***Not- accounted- for DB2 time:*** The DB2 accounting class 2 elapsed time that is not recorded as class 2 CPU time or class 3 suspensions. The most common contributors to this category are:

- MVS paging
- Processor wait time
- On DB2 requester systems, the amount of time waiting for requests to be returned from either VTAM or TCP/IP, including time spent on the network and time spent handling the request in the target or server systems
- Time spent waiting for parallel tasks to complete (when query parallelism is used for the query)

In the DB2 PM accounting report, this information is reported in the field NOT ACCOUNT ( L ).

## A general approach to problem analysis in DB2

The following is a suggested sequence for investigating a response-time problem:

1. If the problem is inside DB2, determine which plan has the longest response time. If the plan can potentially allocate many different packages or DBRMs, determine which packages or DBRMs have the longest response time. Or, if you have a record of past history, determine which transactions show the largest increases.

   Compare class 2 CPU time, class 3 time, and not accounted time. If your performance monitoring tool does not specify times other than Class 2 and Class 3, then you can determine the not accounted for time with the following formula:

   ```
   Other time = Class 2 elapsed time - Class 2 CPU time - Total class 3 time
   ```

2. If the class 2 CPU time is high, investigate by doing the following:
   - Check to see if unnecessary trace options are enabled. Excessive performance tracing can be the reason for a large increase in class 2 CPU time.
   - Check the SQL statement counts on the DB2 PM accounting report. If the profile of the SQL statements has changed significantly, review the application.
   - Use the statistics report to check buffer pool activity, including the buffer pool thresholds. If buffer pool activity has increased, be sure that your buffer pools are properly tuned. For more information on buffer pools, see “Tuning database buffer pools” on page 549.
   - Use EXPLAIN to check the efficiency of the access paths for your application. Based on the EXPLAIN results:
     - Use package-level accounting reports to determine which package or DBRM has a long elapsed time. In addition, use the class 7 CPU time for packages to determine which package or DBRM has the largest CPU time or the greatest increase in CPU time.
     - Use the DB2 PM SQL activity report to analyze specific SQL statements.
     - If you have a history of the performance of the affected application, compare current EXPLAIN output to previous access paths and costs.
     - Check that RUNSTATS statistics are current.
     - Check that databases have been reorganized using the REORG utility.
     - Check which indexes are used and how many columns are accessed. Has your application used an alternative access path because an index was dropped?
     - Examine joins and subqueries for efficiency.

     See “Chapter 33. Using EXPLAIN to improve SQL performance” on page 789 for help in understanding access path selection and analyzing access path problems. DB2 Visual Explain can give you a graphic display on your workstation of your EXPLAIN output.
   - Check the counts in the locking section of the DB2 PM accounting report. If locking activity has increased, see “Chapter 30. Improving concurrency” on page 643. For a more detailed analysis, use the deadlock or timeout traces from statistics trace class 3 and the lock suspension report or trace.

3. If class 3 time is high, check the individual types of suspensions in the “Class 3 Suspensions” section of the DB2 PM accounting report. (The fields referred to here are in Figure 56 on page 529).

   - If LOCK/LATCH ( A ), DRAIN LOCK ( I ), or CLAIM RELEASE ( J ) time is high, see “Chapter 30. Improving concurrency” on page 643.
   - If SYNCHRON. I/O ( B ) time is high, see page 530.
   - If OTHER READ I/O ( D ) time is high, check prefetch I/O operations, disk contention and the tuning of your buffer pools.
   - If OTHER WRITE I/O ( E ) time is high, check the I/O path, disk contention, and the tuning of your buffer pools.
   - If SER.TASK SWTCH ( F ) is high, check open and close activity, as well as commit activity. A high value could also be caused by preformatting data sets for:
     – SYSLGRNG recording service
     – Data set extend/delete/define service

     Consider also, the possibility that DB2 is waiting for Hierarchical Storage Manager (HSM) to recall data sets that had been migrated to tape. The amount of time that DB2 waits during the recall is specified on the RECALL DELAY parameter on installation panel DSNTIPO.

   If accounting class 8 trace was active, each of these suspension times is available on a per-package or per-DBRM basis in the package block of the DB2 PM accounting report.

4. If NOT ACCOUNT. ( L ) time is high, check for paging activity, processor wait time, return wait time for requests to be returned from VTAM or TCP/IP, and wait time for completion of parallel tasks. A high NOT ACCOUNT time is acceptable if it is caused by wait time for completion of parallel tasks.
   - Use RMF reports to analyze paging.
   - Check the SER.TASK SWTCH field in the “Class 3 Suspensions” section of the DB2 PM accounting reports.

Figure 57 on page 535 shows which reports you might use, depending on the nature of the problem, and the order in which to look at them.

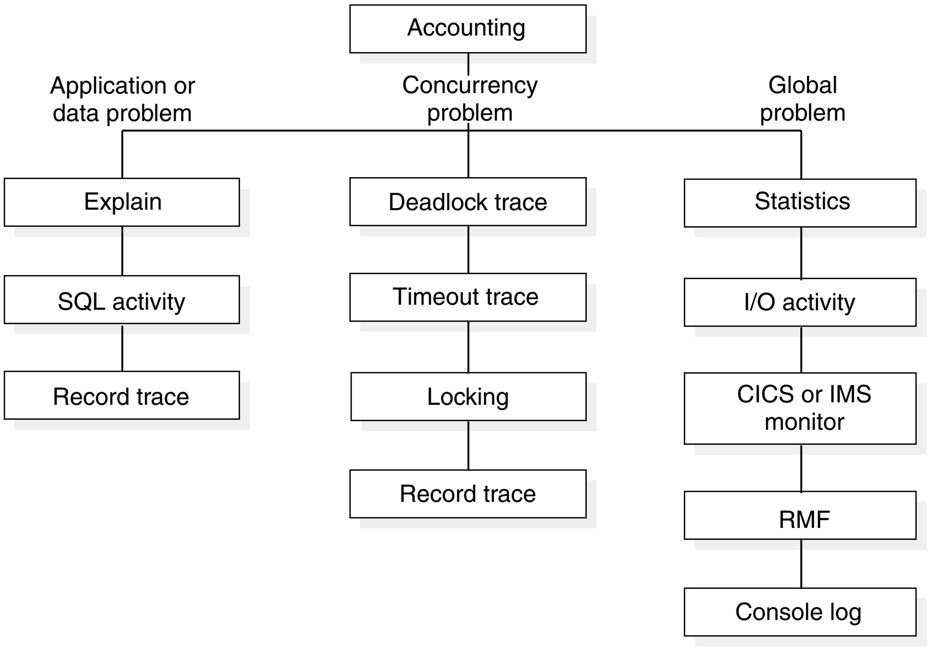


*Figure 57. DB2 PM reports used for problem analysis*

If you suspect that the problem is in DB2, it is often possible to discover its general nature from the accounting reports. You can then analyze the problem in detail based on one of the branches shown in Figure 57:

- Follow the first branch, **Application or data problem**, when you suspect that the problem is in the application itself or in the related data. Also use this path for a further breakdown of the response time when no reason can be identified.
- The second branch, **Concurrency problem**, shows the reports required to investigate a lock contention problem. This is illustrated in “Analyzing a concurrency scenario” on page 702.
- Follow the third branch for a **Global problem**, such as an excessive average elapsed time per I/O. A wide variety of transactions could suffer similar problems.

Before starting the analysis in any of the branches, start the DB2 trace to support the corresponding reports. When starting the DB2 trace:

- Refer to *DB2 PM for OS/390 Report Reference Volume 1* and *DB2 PM for OS/390 Report Reference Volume 2* for the types and classes needed for each report.
- To make the trace data available as soon as an experiment has been carried out, and to avoid flooding the SMF data sets with trace data, use GTF or a user-defined sequential data set as the destination for DB2 performance trace data.

  Alternatively, use DB2 PM’s Collect Report Data function to collect performance data. You specify only the report set, not the DB2 trace types or classes you need for a specific report. Collect Report Data lets you collect data in a TSO data set that is readily available for further processing. No SMF or GTF handling is required.
- To limit the amount of trace data collected, you can restrict the trace to particular plans or users in the reports for SQL activity or locking. However, you cannot so restrict the records for performance class 4, which traces asynchronous I/O for specific page sets. You might want to consider turning on selective traces and be aware of the added costs incurred by tracing.

If the problem is not in DB2, check the appropriate reports from a CICS or IMS reporting tool.

When CICS or IMS reports identify a commit, the time stamp can help you locate the corresponding DB2 PM accounting trace report.

You can match DB2 accounting records with CICS accounting records. If you specify TOKENE=YES on the DSNCRCT macro, the CICS LU 6.2 token is included in the DB2 trace records, in field QWHCTOKN of the correlation header. To help match CICS and DB2 accounting records, specify the option TOKENE=YES or TOKENI=YES in the resource control table. That writes a DB2 accounting record after every transaction. As an alternative, you can produce DB2 PM accounting reports that summarize accounting records by CICS transaction ID. Use the DB2 PM function Correlation Translation to select the subfield containing the CICS transaction ID for reporting.

# Chapter 26. Improving response time and throughput

Response time consists of the following three components:
- Processor resource consumption, which is shown on Figure 56 on page 529 as "CPU TIME".
- Wait time traced in accounting class 3, which includes:
  - I/O wait time (synchronous and asynchronous)
  - Lock and latch wait time
- Other time

In general, you can improve the response time and throughput of your DB2 applications and queries by:
- "Reducing I/O operations"
- "Reducing the time needed to perform I/O operations" on page 541
- "Reducing the amount of processor resources consumed" on page 544

The following sections describe ways to accomplish these goals. The chapter concludes with an overview of how various DB2 response times are reported.

Parallel processing, which is described in "Chapter 34. Parallel operations and query performance" on page 841, can also improve response times. And, DB2 data sharing is also a possible solution for increasing throughput in your system, as well as an opportunity for an improved price for performance ratio. For more information about data sharing, see *DB2 Data Sharing: Planning and Administration*.

## Reducing I/O operations

Reducing the number of I/O operations is one way to improve the response time of your applications and queries. This section describes the following ways you can minimize I/O operations:
- "Use RUNSTATS to keep access path statistics current"
- "Reserve free space in table spaces and indexes" on page 538
- "Make buffer pools large enough for the workload" on page 540
- "Speed up preformatting by allocating in cylinders" on page 540

Using indexes can also minimize I/O operations. For information on indexes and access path selection see "Overview of index access" on page 806.

## Use RUNSTATS to keep access path statistics current

The RUNSTATS utility collects statistics about DB2 objects. These statistics can be stored in the DB2 catalog and are used during the bind process to choose the path in accessing data. If you never use RUNSTATS and subsequently rebind your packages or plans, DB2 will not have the information it needs to choose the most efficient access path. This can result in unnecessary I/O operations and excessive processor consumption. See "Gathering monitor and update statistics" on page 775 for more information on using RUNSTATS.

Run RUNSTATS at least once against each table and its associated indexes. How often you rerun the utility depends on how current you need the catalog data to be. If data characteristics of the table vary significantly over time, you should keep the catalog current with those changes. RUNSTATS is most beneficial for the following:
- Table spaces that contain frequently accessed tables
- Tables involved in a sort

- Tables with many rows
- Tables against which SELECT statements having many search arguments are performed

## Reserve free space in table spaces and indexes

**General-use Programming Interface**

You can use the PCTFREE and FREEPAGE clauses of the CREATE and ALTER TABLESPACE statements and CREATE and ALTER INDEX statements to improve the performance of INSERT and UPDATE operations. The table spaces and indexes for the DB2 catalog can also be altered to modify FREEPAGE and PCTFREE. These options are not applicable for LOB table spaces.

You can change the values of PCTFREE and FREEPAGE for existing indexes and table spaces using the ALTER INDEX and ALTER TABLESPACE statements, but the change has no effect until you load or reorganize the index or table space.

When you specify a sufficient amount of free space, the advantages during normal processing are:
Better clustering of rows (giving faster access)
Fewer overflows
Less frequent reorganizations needed
Less information locked by a page lock
Fewer index page splits

The disadvantages are:
More disk space occupied
Less information transferred per I/O
More pages to scan
Possibly more index levels
Less efficient use of buffer pools and storage controller cache

### Specifying free space on pages

The PCTFREE clause specifies what percentage of each page in a table space or index is left free when loading or reorganizing the data. DB2 uses the free space later on when you insert or update your data; when no free space is available, DB2 holds your additional data on another page. When several records are physically located out of sequence, performance suffers.

The default for PCTFREE for table spaces is 5 (5 percent of the page is free). If you have previously used a large PCTFREE to force one row per page, you should instead use MAXROWS 1 on the CREATE or ALTER TABLESPACE statement. MAXROWS has the advantage of maintaining the free space even when new data is inserted.

The default for indexes is 10. The maximum amount of space that is left free in index nonleaf pages is 10 percent, even if you specify a value higher than 10 for PCTFREE.

To determine the amount of free space currently on a page, run the RUNSTATS utility and examine the PERCACTIVE column of SYSIBM.SYSTABLEPART. See Part 2 of *DB2 Utility Guide and Reference* for information about using RUNSTATS.

## Determining pages of free space

The FREEPAGE clause specifies how often DB2 leaves a full page of free space when loading data or when reorganizing data or indexes. DB2 uses the free space later on when you insert or update your data. For example, if you specify 10 for FREEPAGE, DB2 leaves every 10th page free.

The maximum value you can specify for FREEPAGE is 255; however, in a segmented table space, the maximum value is 1 less than the number of pages specified for SEGSIZE.

## Recommendations for allocating free space

The goal for allocating free space is to maintain the physical clustering of the data and to reduce the need to frequently reorganize table spaces and indexes. However, you do not want to allocate too much disk space.

Use of PCTFREE or FREEPAGE depends on the type of SQL and the distribution of that activity across the table space or index. When deciding whether to allocate free space consider the data and each index separately and assess the insert and update activity on the data and indexes.

***When not to use free space:*** Free space is not necessary if:

- The object is **read-only**.

  If you do not plan to insert or update data in a table, there is no need to leave free space for either the table or its indexes.
- Inserts are at the end.

  For example, if inserts are in ascending order by key of the clustering index or are caused by LOAD RESUME SHRLEVEL NONE, the free space for both the table and clustering index should be zero. Generally, free space is beneficial for a non-clustering index because inserts are usually random. However, if the non-clustering index contains a column with a timestamp value that causes the inserts into the index to be in sequence, the free space should be zero.
- Updates that lengthen varying-length columns are few.

  For example, if you plan only to update fixed-length columns, non-compressed records, the free space for the data should be zero.

***When to use PCTFREE:*** Use PCTFREE if inserted rows are distributed evenly and densely across the key or page range.

If the volume is heavy, use a PCTFREE value greater than the default.

***When to use FREEPAGE:*** Use FREEPAGE if:

- Inserts are concentrated in small areas of the table space or index.

  For indexes where most of the inserts will be random, set FREEPAGE so that when an index split occurs, the new page is often relatively close to the original page. However, if the majority of the inserts occur at the end of the index, set FREEPAGE to 0 to maintain sequential order in the index leaf pages.

  For table spaces, set FREEPAGE so that new data rows can be inserted into a nearby page when the target page is full or locked. A nearby page for a nonsegmented table space is within 16 pages on either side of the target page. For a segmented table space, a nearby page is within the same segment as the target page.
- MAXROWS 1 or rows are larger than half a page, because you cannot insert a second row on a page.

***Additional recommendations:***

- For concurrency, use MAXROWS or larger PCTFREE values for small tables and shared table spaces that use page locking. This reduces the number of rows per page, thus reducing the frequency that any given page is accessed.
- For the DB2 catalog table spaces and indexes, use the defaults for PCTFREE. If additional free space is needed, use FREEPAGE.

**End of General-use Programming Interface**

## Make buffer pools large enough for the workload

Make buffer pools as large as you can afford, because:

- It might mean fewer I/O operations and therefore faster access to your data.
- It can reduce I/O contention for the most frequently used tables and indexes.
- It can speed sorting by reducing I/O contention for work files.

However, there are many factors to consider when determining how many buffer pools to have and how big they should be. See "Determining size and number of buffer pools" on page 560 for more information.

## Speed up preformatting by allocating in cylinders

This section describes a general way to speed up preformatting of data by allocating in cylinders, and a more specific way you can preformat a table space before inserting data.

### Allocate space in cylinders

Specify your space allocation amounts to ensure allocation by CYLINDER. This can reduce the time required to do SQL mass inserts and to recover a table space from the log; it does not affect the time required to recover a table space from an image copy or to run the REBUILD utility.

When inserting records, DB2 preformats space within a page set as needed. The allocation amount, which is either CYLINDER or TRACK, determines the amount of space that is preformatted at any one time. See "Preformatting during LOAD" for a way you can preformat data using LOAD or REORG.

Because less space is preformatted at one time for the TRACK allocation amount, a mass insert can take longer when the allocation amount is TRACK than the same insert when the allocation amount is CYLINDER.

The allocation amount is dependent on device type and the number of bytes you specify for PRIQTY and SECQTY when you define table spaces and indexes. The default SECQTY is 10 percent of the PRIQTY, or 3 times the page size, whichever is larger. This default quantity is an efficient use of storage allocation. Choosing a SECQTY value that is too small in relation to the PRIQTY value results in track allocation.

For more information about how space allocation amounts are determined, see the description of the DEFINE CLUSTER command in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

### Preformatting during LOAD

When DB2's preformatting delays impact the performance or execution time consistency of applications that do heavy insert processing, and if the table size can be predicted for a business processing cycle, consider using the PREFORMAT option of LOAD and REORG. If you preformat during LOAD or REORG, DB2 does not have to preformat new pages during execution. When the preformatted space is used and when DB2 has to extend the table space, normal data set extending and preformatting occurs.

Consider preformatting only if preformatting is causing a measurable delay with the insert processing or causing inconsistent elapsed times for insert applications. For more information about the PREFORMAT option, see Part 2 of *DB2 Utility Guide and Reference*.

**Recommendation:** Quantify the results of preformatting in your environment by assessing the performance both before and after using preformatting.

## Reducing the time needed to perform I/O operations

You can reduce the time needed to perform individual I/O operations in several ways:
- Create additional work file table spaces
- “Distribute data sets efficiently” on page 542
- “Ensure sufficient primary allocation quantity” on page 544

For information on parallel operations, see “Chapter 34. Parallel operations and query performance” on page 841.

For information on I/O scheduling priority, see “MVS performance options for DB2” on page 614.

### Create additional work file table spaces

If your applications require any of the following, allocate additional work file table spaces on separate disk volumes in a work file database (database DSNDB07 in a non data-sharing environment) to help minimize I/O contention:
- Large concurrent sorts or a single large sort (especially of table spaces defined as LARGE)
- Created temporary tables
- Star joins
- Non-correlated subqueries
- Materialized views

For a single query, the recommendation for the number of work file disk volumes is to have whichever is more:
- Five
- One-fifth the maximum number of data partitions

For concurrently runnng queries, multiply this value by the number of concurrent queries.

In addition, in a query parallelism environment, the number of work file disk volumes should be at least equal to the maximum number of parallel operations that is seen for queries in the given workload.

Place these volumes on different channel or control unit paths. Monitor the I/O activity for the work file table spaces, because you might need to further separate this work file activity to avoid contention. As the amount of work file activity increases, consider increasing the size of the buffer pool for work files to support concurrent activities more efficiently. The general recommendation for the work file buffer pool is to increase the size to minimize the following buffer pool statistics:

- MERGE PASSES DEGRADED, which should be less than 1% of MERGE PASS REQUESTED
- WORKFILE REQUESTS REJECTED, which should be less than 1% of WORKFILE REQUEST ALL MERGE PASSES
- Synchronous read I/O, which should be less than 1% of pages read by prefetch
- Prefetch quantity of 4 or less, which should be near 8

During the installation or migration process, you allocated table spaces for 4KB buffering, and for 32KB buffering. To create additional work file table spaces, use SQL statements similar to those in job DSNTIJTM.

***Steps to create a work file table space:*** Use the following steps to create a new work file table space, *xyz*. (If you are using DB2-managed data sets, omit the step to create the data sets.)

1. Define the required data sets using the VSAM DEFINE CLUSTER statement before creating the table space. You must specify a minimum of 26 4KB pages for the work file table space. For more information on the size of sort work files see “Understanding how sort work files are allocated” on page 575. See also Figure 3 on page 36 for more information on the DEFINE CLUSTER statement.
2. Issue the following command to stop all current users of the work file database:
   ```
   -STOP DATABASE (DSNDB07)
   ```
3. Enter the following SQL statement:
   ```
   CREATE TABLESPACE xyz IN DSNDB07
      BUFFERPOOL BP0
      CLOSE NO
      USING VCAT DSNC710;
   ```
4. Enter the following command:
   ```
   -START DATABASE (DSNDB07)
   ```

## Distribute data sets efficiently

Avoid I/O contention and increase throughput through the I/O subsystem by placing frequently used data sets on fast disk devices and by distributing I/O activity. Distributing I/O activity is less important when you use disk devices with Parallel Access Volumes (PAV) support and multiple allegiance support. (For more information, see “Parallel Access Volumes (PAV)” on page 613 and “Multiple Allegiance” on page 613.)

### Put frequently used data sets on fast devices

Assign the most frequently used data sets to the faster disk devices at your disposal. For partitioned table spaces, you might choose to have some partitions on faster devices than other partitions. Placing frequently used data sets on fast disk devices also improves performance for nonpartitioned table spaces. You might consider partitioning any nonpartitioned table spaces that have excessive I/O contention at the data set level.

### Distribute the I/O

Allocate frequently used data sets or partitions across your available disk volumes so that I/O operations are distributed. Even with RAID devices, in which the data set is spread across the physical disks in an array, it is important that is accessed at the same time on separate logical volumes to reduce the chance of an I/O request being queued in MVS.

Consider isolating data sets with characteristics that do not complement other data sets. For example, do not put high volume transaction work that uses synchronous reads on the same volume as something of lower importance that uses list prefetch.

***Consider the partitioning scheme:*** If it is critical that partitions of your partitioned table spaces be of relatively the same size (which can be a great benefit for query parallelism), consider using a ROWID column as all or part of the partitioning key. For partitions that are of unequal size to such an extent that they are negatively affecting performance, alter the partitioning index limiting key values and then reorganize the affected partitions to rebalance the data.

***Spread data sets of nonpartitioning indexes:***

**General-use Programming Interface**

If I/O contention on a nonpartitioning index has prevented you from running batch update jobs in parallel, use the PIECESIZE option of CREATE or ALTER INDEX to indicate how large DB2 should make the data sets that make up a nonpartitioning index. As the specification of the maximum addressability of a data set, the piece size of an index limits how much data DB2 puts into a data set before it is broken into multiple pieces (data sets). By making the piece size smaller than the default value, for example, you can end up with many more data sets. If you spread these data sets across the available I/O paths, you can reduce the physical contention on the nonpartitioning index.

***Choosing a value for PIECESIZE:***To choose a PIECESIZE value, divide the size of the nonpartitioning index by the number of data sets that you want. For example, to ensure that you have 5 data sets for the nonpartitioning index, and your nonpartitioning index is 10 MB (and not likely to grow much), specify PIECESIZE 2M. If your nonpartitioning index is likely to grow, choose a larger value.

When choosing a value, remember that the maximum partition size of the table space determines the maximum number of data sets that the index can use. If the underlying table space is defined with a DSSIZE of 4G or greater (or with LARGE), the limit is 254 pieces; otherwise, the limit is 32 pieces. Nonpartitioning indexes that were created on LARGE table spaces in Version 5 and migrated to Version 7 can have only 128 pieces. If an attempt is made to allocate more data sets than the limit, an abend occurs.

Keep your PIECESIZE value in mind when you are choosing values for primary and secondary quantities. Ideally, although PIECESIZE has no effect on primary and secondary space allocation, the value of your primary quantity and the secondary quantities should be evenly divisible into PIECESIZE to avoid wasting space. Because the underlying data sets are always allocated at the size of PRIQTY and extended, when possible, with the size of SECQTY, understand the implications of their values with the PIECESIZE value:

- If PRIQTY is larger than PIECESIZE, a new data set is allocated and used when the file size exceeds PIECESIZE. Thus, part of the allocated primary storage goes unused, and no secondary extents are created.
- If PRIQTY is smaller than PIECESIZE and SECQTY is not zero, secondary extents are created until the total file size equals or exceeds PIECESIZE. After the allocation of a secondary extent causes the total file size to meet or exceed PIECESIZE, a new data set is allocated and used. When the total file size exceeds PIECESIZE, the part of secondary storage that is allocated beyond PIECESIZE goes unused.

- If PRIQTY is smaller than PIECESIZE and SECQTY is zero, an ″unavailable resource″ message is returned when the data set fills up. No secondary extents are created nor are additional data sets allocated.

***Identifying suitable indexes:*** Any secondary index that has a lot of I/O and a high IOS queue time is a good candidate for breaking up into smaller pieces. Use the statistics trace to identify I/O intensive data sets. IFCID 199 contains information about every data set that averages more than one I/O per second during the statistics interval. IOS queue time that is 2 or 3 times higher than connect time is considered high. The RMF (Resource Measurement Facility) Device Activity report provides IOS time and CONN time.

***Determining the number of pieces an index is using:*** You can use one of the following techniques to determine the number of pieces that an index uses:

- For DB2-managed data sets, use access method services LISTCAT to check the number of data sets that have been created.
- For user-managed data sets, examine the high-used RBA (HURBA) for each data set.

**End of General-use Programming Interface**

## Ensure sufficient primary allocation quantity

Specifying sufficient primary allocation for frequently used data sets minimizes I/O time, because the data is not physically located at different places on the disks.

It can be helpful to list the VTOC occasionally to determine the number of secondary allocations that have been made for your more frequently used data sets. Or, you can use IFCID 0258 in the statistics class 3 trace to monitor data set extensions.

If you discover that the data sets backing frequently used table spaces or indexes have an excessive number of extents, and if the data sets are user-defined, you can use access method services to reallocate the affected data sets using a larger primary allocation quantity. If the data sets were created using STOGROUPs, you can use the procedure for modifying the definition of table spaces presented in “Altering table spaces” on page 57.

***Specify primary quantity for nonpartitioning indexes:*** To prevent wasted space for nonpartitioning indexes, make sure that the value of PRIQTY + (*N* × SECQTY) is a value that evenly divides into PIECESIZE. For more information about PIECESIZE, see Chapter 5 of *DB2 SQL Reference*.

## Reducing the amount of processor resources consumed

Many factors affect the amount of processor resources that DB2 consumes. This section describes ways to reduce DB2 consumption of these resources.

- Reuse threads for your high-volume transactions
- “Minimize the use of DB2 traces” on page 545
- “Use fixed-length records” on page 546

Consider also caching authorizations for plans, packages, and routines (user-defined functions and stored procedures). See “Caching authorization IDs for best performance” on page 120 for more information.

## Reuse threads for your high-volume transactions

For high volume transactions, reusing threads can help performance significantly.

- For IMS, process multiple input messages in one scheduling of the IMS processing program by setting PROCLIM to a value greater than 1 and using class priority scheduling. This shares the cost of thread creation and termination among more than one transaction. Alternatively, you can reuse threads with wait for input (WFI), or the IMS fast path and class scheduling. See Part 2 of *DB2 Installation Guide* for more information.
- For CICS, you can enhance thread reuse through specifications for pool and entry threads in the RCT. Consider using protected entry threads for high volume transactions. See "CICS design options" on page 633 for details.
- If you are using the Recoverable Resource Manager Services attachment facility, see the RRSAF chapter of Part 6 of *DB2 Application Programming and SQL Guide* for more information about reusing threads.

## Minimize the use of DB2 traces

Using the DB2 trace facility, particularly performance and global trace, can consume a large amount of processing resources. Suppressing these trace options significantly reduces additional processing costs.

### Global trace

Global trace requires 20 percent to 100 percent additional processor utilization. If conditions permit at your site, the DB2 global trace should be turned off. You can do this by specifying NO for the field TRACE AUTO START on panel DSNTIPN at installation. Then, if the global trace is needed for serviceability, you can start it using the START TRACE command.

### Accounting and statistics traces

Enabling accounting class 2 along with accounting classes 1 and 3 provides additional detail relating directly to the accounting record IFCID 0003, as well as recording thread level entry into and exit from DB2. This allows you to separate DB2 times from application times. Running accounting class 2 does add to the cost of processing. How much overhead occurs depends on how much SQL the application issues. Typically, an online transaction incurs an additional 2.5 percent when running with accounting class 2. A typical batch query application, which accesses DB2 more often, incurs about 10 percent overhead when running with accounting class 2. If most of your work is through CICS, you most likely do not need to run with class 2, because the class 1 and class 2 times are very close.

If you have very light DB2 usage and you are using Measured Usage, then you need the SMF 89 records. In other situations, be sure that SMF 89 records are not recorded to avoid this overhead.

### Audit trace

The performance impact of auditing is directly dependent on the amount of audit data produced. When the audit trace is active, the more tables that are audited and the more transactions that access them, the greater the performance impact. The overhead of audit trace is typically less than 5 percent.

When estimating the performance impact of the audit trace, consider the frequency of certain events. For example, security violations are not as frequent as table accesses. The frequency of utility runs is likely to be measured in executions per day. On the other hand, authorization changes can be numerous in a transaction environment.

### Performance trace

Consider turning on only the performance trace classes required to address a specific performance problem. The combined overhead of all performance classes runs from about 20 percent to 100 percent.

The overhead for performance trace classes 1 through 3 is typically in the range of 5 percent to 30 percent.

Suppressing the IRLM, MVS, IMS, and CICS trace options also reduces overhead.

## Use fixed-length records

Use fixed-length columns rather than varying-length columns, particularly in tables that contain many columns. This can reduce processor use, but is offset by the need for more disk space. If you must use varying-length columns, see Table 84 on page 605 for recommendations about where to place those columns for the best performance and to reduce logging.

If you use ALTER to add a fixed-length column to a table, that column is treated as variable-length until the table has been reorganized.

## Understanding response time reporting

To correctly monitor response time, you must understand how it is reported. Response time can be measured in several different ways. Figure 58 on page 547 shows how some of the main measures relate to the flow of a transaction.

In Figure 58 on page 547, the following times can be distinguished:

**End user response time**

This is the time from the moment the end user presses the enter key until he or she receives the first response back at the terminal.

**DB2 accounting elapsed times**

These times are collected in the records from the accounting trace and can be found in the DB2 PM accounting reports. They are taken over the accounting interval between the point where DB2 starts to execute the first SQL statement, and the point preceding thread termination or reuse by a different user (sign-on).

This interval excludes the time spent creating a thread, and it includes a portion of the time spent terminating a thread.

For parallelism, there are special considerations for doing accounting. See “Monitoring parallel operations” on page 850 for more information.

Elapsed times for stored procedures or user-defined functions separate the time spent in the allied address space and the time spent in the stored procedures address space.

There are two elapsed times:

– Class 1 elapsed time

  This time is always presented in the accounting record and shows the duration of the accounting interval. It includes time spent in DB2 as well as time spent in the front end. In the accounting reports, it is referred to as “application time.”

– Class 2 elapsed time

  Class 2 elapsed time, produced only if the accounting class 2 is active, counts only the time spent in the DB2 address space during the accounting interval. It represents the sum of the times from any entry into DB2 until the corresponding exit from DB2. It is also referred to as the time spent in DB2.

If class 2 is not active for the duration of the thread, the class 2 elapsed time does not reflect the entire DB2 time for the thread, but only the time when the class was active.

**DB2 total transit time**

In the particular case of an SQL transaction or query, the “total transit time” is the elapsed time from the beginning of create thread, or sign-on of another authorization ID when reusing the thread, until either the end of the thread termination, or the sign-on of another authorization ID.

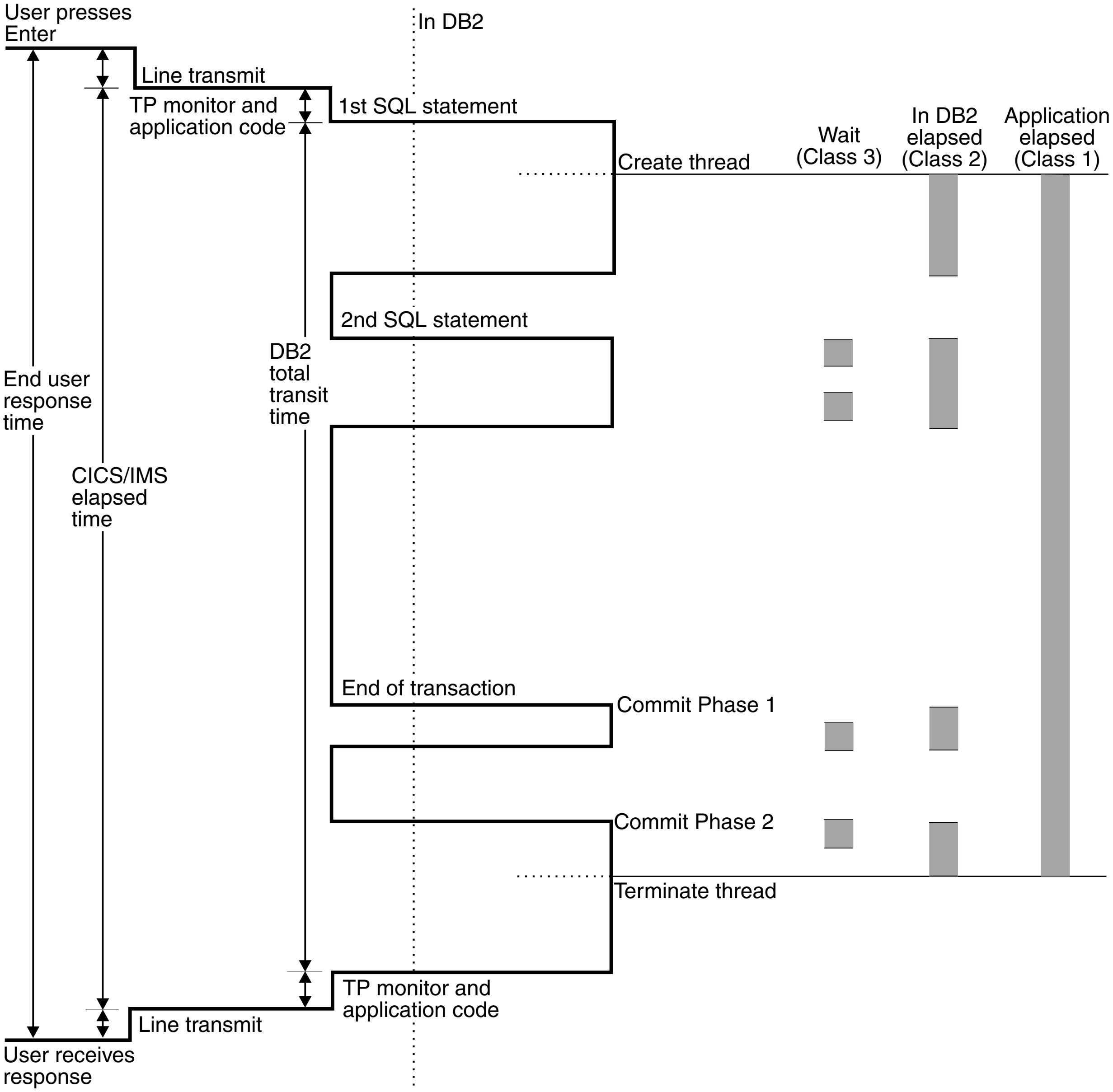


*Figure 58. Transaction response times. Class 1 is standard accounting data. Class 2 is elapsed and processor time in DB2. Class 3 is elapsed wait time in DB2. Standard accounting data is provided in IFCID 0003, which is turned on with accounting class 1. When accounting classes 2 and 3 are turned on as well, IFCID 0003 contains additional information about DB2 times and wait times.*

# Chapter 27. Tuning DB2 buffer, EDM, RID, and sort pools

Proper tuning of your virtual buffer pools, EDM pools, RID pools, and sort pools can improve the response time and throughput for your applications and provide optimum resource utilization. Using data compression can also improve buffer pool hit ratios and reduce table space I/O rates. For more information on compression, see “Compressing your data” on page 606. This chapter covers the following topics:

- “Tuning database buffer pools”
- “Tuning the EDM pool” on page 570
- “Increasing RID pool size” on page 574
- “Controlling sort pool size and sort processing” on page 574

## Tuning database buffer pools

Buffer pools are areas of virtual storage that temporarily store pages of table spaces or indexes. When an application program accesses a row of a table, DB2 places the page that contains that row in a buffer. If the requested data is already in a buffer, the application program does not have to wait for it to be retrieved from disk. Avoiding the need to retrieve data from disk results in faster performance.

If the row is changed, the data in the buffer must be written back to disk eventually. But that write operation might be delayed until DB2 takes a checkpoint, or until one of the related write thresholds is reached. (In a data sharing environment, however, the writing mechanism is somewhat different. See Chapter 6 of *DB2 Data Sharing: Planning and Administration* for more information.)

The data remains in the buffer until DB2 decides to use the space for another page. Until that time, the data can be read or changed without a disk I/O operation.

DB2 allows you to use up to 50 buffer pools that contain 4-KB buffers and up to 10 buffer pools each for 8-KB, 16-KB, and 32-KB buffers. You can set the size of each of those buffer pools separately when installing DB2. You can change the sizes and other characteristics of a buffer pool at any time while DB2 is running, by using the ALTER BUFFERPOOL command.

This section includes the following topics:

- “Buffer pools and hiperpools” on page 550
- “Buffer pools and data spaces” on page 552
- “Terminology: Types of buffer pool pages” on page 553
- “Read operations” on page 554
- “Write operations” on page 554
- “Assigning a table space or index to a virtual buffer pool” on page 555
- “Buffer pool thresholds” on page 555
- “Determining size and number of buffer pools” on page 560
- “Choosing a page-stealing algorithm” on page 562
- “Monitoring and tuning buffer pools using online commands” on page 563
- “Using DB2 PM to monitor buffer pool statistics” on page 567

***Buffer Pool Tool:*** You can use the Buffer Pool Tool feature of DB2 to do “what if” analysis of your buffer pools.

## Choose backing storage: primary or data space

You have several ways to configure each buffer pool's backing storage:

- Keep the buffer pool strictly in ssnmDBM1 address space. This buffer pool is a *primary* virtual pool to distinguish it from buffer pools in data spaces. This option performs well, but because it can have a significant impact on your storage, you might need to look at the other available options for reducing the impact on that storage.
- Extend the primary pool to hiperspace, which means the primary buffer pool has space in *ssnm*DBM1 and in hiperspace, called the hiperpool. See "Buffer pools and hiperpools" for information about this option.
- Keep the buffer pool in a *data space*. See "Buffer pools and data spaces" on page 552 for more information about this option.

Use the virtual pool type (VPTYPE) field of installation panels DSNTIP1, 2, and 6 to specify where buffer pools should reside. A value of P indicates that the buffer pool resides in DB2's primary address space (ssnmDBM1). If you want to extend the primary pool to hiperspace, specify a non-zero value in the Hiperpool field of the panel.

Specify D for VPTYPE if you want the virtual buffer pool to reside in a data space.

***Altering VPTYPE:*** You can use the ALTER BUFFERPOOL command to change the VPTYPE, but this change requires a reallocation of the buffer pool. The procedures for changing the VPTYPE are described with the command in Chapter 2 of *DB2 Command Reference*.

### Buffer pools and hiperpools

If your DB2 subsystem is on a processor that has the Fast Sync data mover facility (such as an S/390 G5/G6 enterprise server) or that has the Asynchronous Data Mover hardware feature installed, you can use hiperspaces to extend DB2's virtual buffer pools. A *hiperspace* is a storage space of up to 2 GB that a program can use as a data buffer. A hiperspace is addressable in 4 KB blocks; in other words, it is page addressable. You cannot put a primary buffer pool into both a hiperpool and data space. For more information on hiperspace, see *OS/390 MVS Programming: Extended Addressability Guide*.

DB2 cannot directly manipulate data that resides in hiperspace. But, it can transfer the data from hiperspace into a virtual buffer pool much faster than it could get it from disk. To distinguish between hiperpools and buffer pools, remember that regular DB2 buffer pools are called virtual buffer pools.

***Two levels of storage:*** When you choose to use hiperpools, DB2 maintains two levels of storage for each buffer pool:

- The first level of storage, the virtual buffer pool, is allocated from DB2's *ssnm*DBM1 address space. A virtual buffer pool is backed by central storage, expanded storage, or auxiliary storage. The sum of all DB2 virtual buffer pools cannot exceed 1.6 GB.
- The second level of storage, the hiperpool, uses the MVS hiperspace facility to utilize expanded storage only (ESO) hiperspace. The sum of all hiperpools cannot exceed 8 GB.

A hiperpool is an extension to a virtual buffer pool and must always be associated with a virtual buffer pool. You can define a hiperpool to be larger than its corresponding virtual buffer pool. Figure 59 illustrates the relationship between a virtual buffer pool and its corresponding hiperpool.

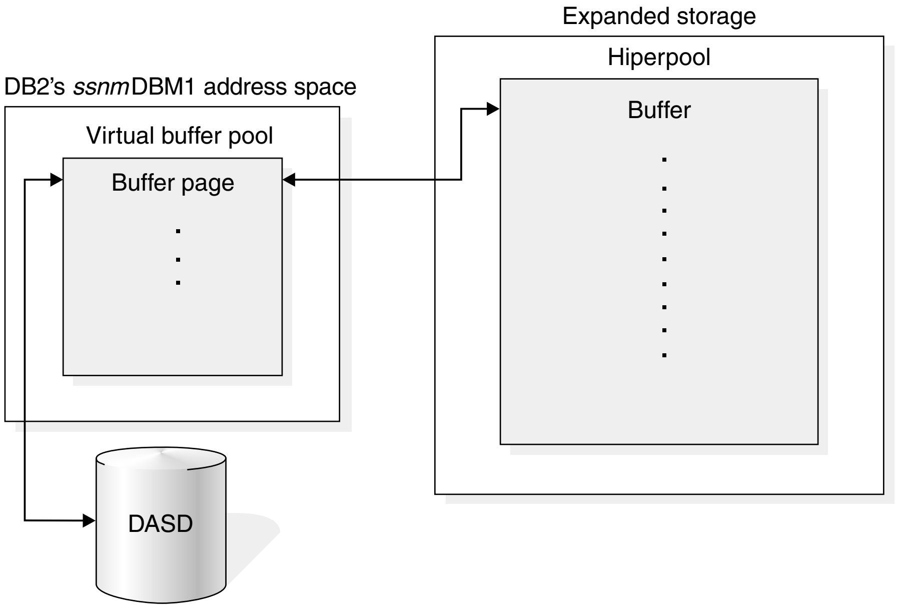


*Figure 59. Relationship between virtual buffer pool and hiperpool*

Reducing the size of your virtual buffer pools and allocating hiperpools provides better control over the use of central storage and can reduce overall contention for central storage.

A virtual buffer pool and its corresponding hiperpool, if defined, are built dynamically when the first page set that references those buffer pools is opened.

***Advantages of hiperpools:*** Virtual buffer pools hold the most frequently accessed data, while hiperpools serve as a cache for data that is accessed less frequently. When a row of data is needed from a page in a hiperpool, the entire page is read into the corresponding virtual buffer pool. If the row is changed, the page is **not** written back to the hiperpool until it has been written to disk: all read and write operations to data in the page, and all disk I/O operations, take place in the virtual buffer pool. The hiperpool holds only pages that have been read into the virtual buffer pool and might have been discarded; they are kept in case they are needed again.

Because read operations from disk are not required to access data that resides in hiperspace, response time is shorter than for disk retrieval. Retrieving pages that are cached in hiperpools takes only microseconds, rather than the milliseconds needed for retrieving a page from disk, which reduces transaction and query response time.

***The good storage citizen: using the CASTOUT attribute:*** Because expanded storage is a shared system resource, DB2 is not the only user of your MVS system's expanded storage. If DB2 monopolizes the available hiperspace, performance could be adversely affected. The CASTOUT option of ALTER BUFFERPOOL gives you some control over DB2's use of hiperspace.

If you specify CASTOUT as YES, your MVS system can steal, or remove, pages from the hiperpool when the need for expanded storage arises and usage of the hiperpool is low. A stolen page is no longer available to DB2; the data will need to be retrieved from disk when next referenced. For that reason, a page brought in from the hiperpool and updated in the virtual buffer pool cannot be written back to the hiperpool unless it is first written to disk.

Specifying CASTOUT as NO tells MVS to give high priority to keeping the data cached in the hiperpool. CASTOUT(NO) places a heavy demand on expanded storage. In general, specify NO to improve response time in only your most critical applications. For example, it is possible to keep an entire index or table in hiperspace almost constantly, by assigning it to a virtual buffer pool whose hiperpool has CASTOUT as NO. Access to those pages is fast, but they might take up a significant proportion of the available expanded storage.

**Recommendation:** Choose CASTOUT (YES).

## Buffer pools and data spaces

Another option to consider for some of your buffer pools is to have DB2 put them in *data spaces*. Like hiperspaces, data spaces are data-only spaces; that is, no program code can run in those areas. With data spaces, though, the system uses the same resources to back data space virtual storage as it uses to back address space virtual storage: a combination of central storage and expanded storage frames (if available), and auxiliary storage slots. The system can move low-use pages of data space storage to auxiliary storage and bring them in again. The paging activity for a data space includes I/O between auxiliary storage paging devices and central storage.

Figure 60 shows DB2 using a data space for a virtual buffer pool.

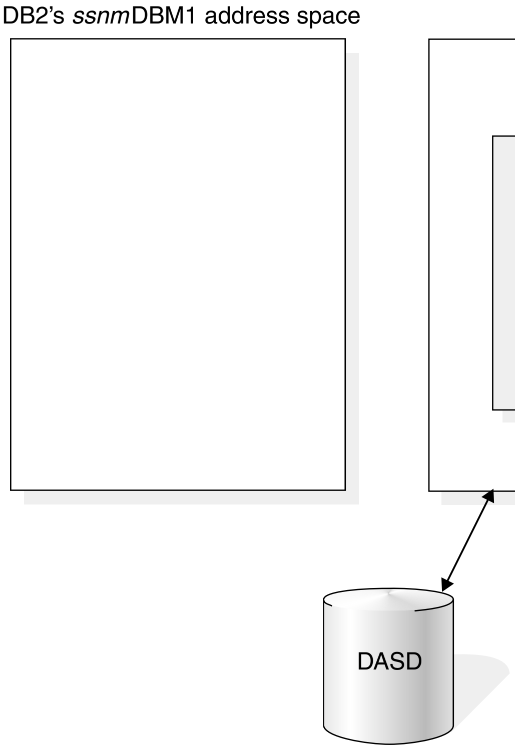


*Figure 60. Using a data space for DB2 virtual buffer pools*

As explained in “Advantages of data spaces”, your DB2 subsystem should run on a processor that has enough real memory to back the data space buffer pools to achieve the full benefits of using data spaces.

For more information about data spaces, see *OS/390 MVS Programming: Extended Addressability Guide*.

***Storage limits for data spaces:*** Each data space can accommodate almost 2 GB worth of buffers and any single buffer pool can span multiple data spaces. The sum of all data space buffers cannot exceed 8 million. This translates to the maximum sizes described in Table 73:

*Table 73. Maximum amount of storage available for data space buffers*

| **If all buffers are this size...** | **The total amount of data space storage is...** |
|---|---|
| 4 KB | 32 GB |
| 8 KB | 64 GB |
| 16 KB | 128 GB |
| 32 KB | 256 GB |

***Total storage in the ssnmDBM1 address space:*** Each buffer in a data space requires about 128 bytes of storage in DB2’s *ssnm*DBM1 address space. DB2 does not allow more than 1.6 GB of storage in *ssnm*DBM1 address space for virtual pool buffers and data space buffer control storage. Message DSNB508I is issued if the amount of space exceeds 1.6 GB.

***Advantages of data spaces:*** With the IBM @server zSeries 900 (z900) along with OS/390 Version 2 Release 10 or z/OS Version 1 Release 1 64-bit real storage support, you can use data space buffer pools to gain significant performance advantages by allowing you to configure larger buffer pools and to relieve storage constraints in DB2’s *ssnm*DBM1 address space. A large data space buffer pool configuration has the following advantages over a similarly sized virtual buffer pool and hiperpool configuration:

- DB2 can put changed pages in data spaces. (Pages in hiperpools must be clean.)
- DB2 can do I/O directly in and out of a data space but not a hiperpool.
- Internal latching and unlatching and LRU management occurs much less frequently. Latching overhead and LRU management can be a concern when pages are moved frequently between a virtual buffer pool and its associated hiperpool.
- The maximum size for data space buffer pools is larger, as described in “Storage limits for data spaces”.
- Less *ssnm*DBM1 storage is used for a data space virtual pool when compared with a primary space virtual pool with its associated hiperpool.

If your DB2 subsystem does not run on a z900 server, the main reason to choose data spaces is to relieve storage constraints in DB2’s *ssnm*DBM1 address space (hiperpools can also be used for this purpose). Otherwise, the use of data spaces provides no immediate benefit.

## Terminology: Types of buffer pool pages

At any moment, a database virtual buffer pool can have three types of pages:

***In-use pages:*** These are pages that are currently being read or updated. The data they contain is available for use by other applications.

***Updated pages:*** These are pages whose data has been changed but have not yet been written to disk. After the updated page has been written to disk, it remains in the virtual buffer pool available for migration to the corresponding hiperpool. In this case, the page is not considered to be “updated” until it is changed again.

***Available pages:*** These pages can be considered for new use, to be overwritten by an incoming page of new data. Both in-use pages and updated pages are *unavailable* in this sense; they are not considered for new use.

## Read operations

DB2 uses three read mechanisms: *normal read*, *sequential prefetch*, and *list sequential prefetch*.

***Normal read:*** Normal read is used when just one or a few consecutive pages are retrieved. The unit of transfer for a normal read is one page.

***Sequential prefetch:*** Sequential prefetch is performed concurrently with other operations of the originating application program. It brings pages into the virtual buffer pool before they are required and reads several pages with a single I/O operation.

Sequential prefetch can be used to read data pages, by table space scans or index scans with clustered data reference. It can also be used to read index pages in an index scan. Sequential prefetch allows CP and I/O operations to be overlapped.

See “Sequential prefetch (PREFETCH=S)” on page 824 for a complete description of sequential prefetch.

***List sequential prefetch:*** List sequential prefetch is used to prefetch data pages that are not contiguous (such as through non-clustered indexes). List prefetch can also be used by incremental image copy. For a complete description of the mechanism, see “List prefetch (PREFETCH=L)” on page 825.

## Write operations

Write operations are usually performed concurrently with user requests. Updated pages are queued by data set until they are written when:

- A checkpoint is taken.
- The percentage of updated pages in a virtual buffer pool for a single data set exceeds a preset limit called the vertical deferred write threshold (VDWQT). For more information on this threshold, see “Buffer pool thresholds” on page 555.
- The percentage of unavailable pages in a virtual buffer pool exceeds a preset limit called the deferred write threshold (DWQT). For more information on this threshold, see “Buffer pool thresholds” on page 555.

Table 74 lists how many pages DB2 can write in a single I/O operation.

*Table 74. Number of pages that DB2 can write in a single I/O operation*

| Page size | Number of pages |
|---|---|
| 4 KB | 32 |
| 8 KB | 16 |

Table 74. Number of pages that DB2 can write in a single I/O operation (continued)

| Page size | Number of pages |
|---|---|
| 16 KB | 8 |
| 32 KB | 4 |

# Assigning a table space or index to a virtual buffer pool

How you assign data to buffer pools can have a significant impact on performance. See “Reasons to choose more than one buffer pool” on page 562 for guidance in choosing a scheme for assigning data and indexes to buffer pools.

## Assigning data to default buffer pools

Installation panel DSNTIP1 lets you set one default buffer pool for user data and one for user indexes. It is a good idea to choose values other than the defaults, BP0 for both, for these options, because BP0 must be used by the DB2 catalog and directory. BP0 is much more difficult to monitor and tune if user data and indexes also use that buffer pool.

## Assigning data to particular buffer pools

You assign a table space or an index to a particular virtual buffer pool by a clause of the following SQL statements: CREATE TABLESPACE, ALTER TABLESPACE, CREATE INDEX, ALTER INDEX. The virtual buffer pool is actually allocated the first time a table space or index assigned to it is opened.

***BP0 default size:*** You cannot use the ALTER statement to change the assignment of the catalog and directory; they are always assigned to BP0. BP0 is also the default buffer pool for sorting, but you can change that by assigning the work file table spaces to another buffer pool. BP0 has a default size of 2000 buffers, and a minimum of 56 buffers. As with any other buffer pool, you can change the size using the ALTER BUFFERPOOL command.

# Buffer pool thresholds

The information under this heading, up to “Determining size and number of buffer pools” on page 560, is General-use Programming Interface and Associated Guidance Information as defined in “Notices” on page 1095.

DB2’s use of a virtual buffer pool or hiperpool is governed by several preset values called *thresholds*. Each threshold is a level of use which, when exceeded, causes DB2 to take some action. When you reach some thresholds, it indicates a problem, while reaching other thresholds merely indicates normal buffer management. The level of use is usually expressed as a percentage of the total size of the virtual buffer pool or hiperpool. For example, the “immediate write threshold” of a virtual buffer pool (described in more detail later) is set at 97.5%; when the percentage of unavailable pages in a virtual buffer pool exceeds that value, DB2 writes pages to disk when updates are completed.

Figure 61 on page 556 shows the relationship between some of the virtual buffer pool thresholds and the updated, in-use, and available pages.

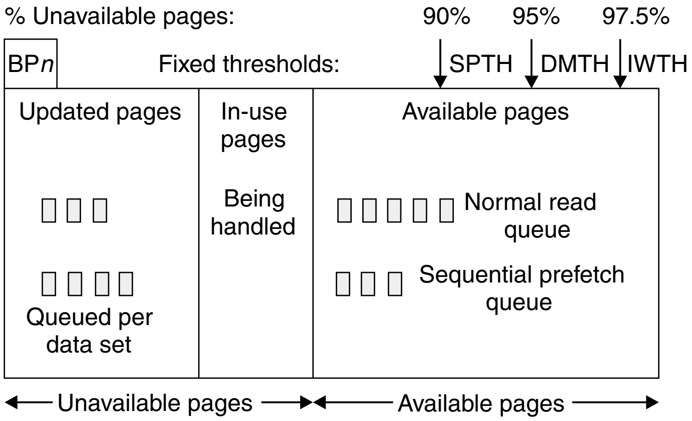


*Figure 61. Database virtual buffer pool. SPTH, DMTH, and IWTH are the performance critical thresholds.*

***Thresholds for very small buffer pools:*** This section describes fixed and variable thresholds that are in effect for buffer pools that are sized for the best performance; that is, for buffer pools of 1000 buffers or more. For very small buffer pools, some of the thresholds are lower to prevent "buffer pool full" conditions, but those thresholds are not described.

## Fixed thresholds

Some thresholds, like the immediate write threshold, you cannot change. Monitoring buffer pool usage includes noting how often those thresholds are reached. If they are reached too often, the remedy is to increase the size of the virtual buffer pool, which you can do with the ALTER BUFFERPOOL command. Increasing the size, though, can affect other buffer pools, depending on the total amount of central and expanded storage available for your buffers.

The fixed thresholds are more critical for performance than the variable thresholds. Generally, you want to set virtual buffer pool sizes large enough to avoid reaching any of these thresholds, except occasionally.

Each of the fixed thresholds is expressed as a percentage of the buffer pool that might be occupied by unavailable pages.

The fixed thresholds are (from highest to lowest value):

- **Immediate write threshold (IWTH)—97.5%**

  This threshold is checked whenever a page is to be updated. If it has been exceeded, the updated page is written to disk as soon as the update completes. The write is synchronous with the SQL request; that is, the request waits until the write has been completed and the two operations are not carried out concurrently.

  Reaching this threshold has a significant effect on processor usage and I/O resource consumption. For example, updating three rows per page in 10 sequential pages ordinarily requires one or two write operations. When IWTH is exceeded, however, the updates require 30 synchronous writes.

  Sometimes DB2 uses synchronous writes even when the IWTH is not exceeded; for example, when more than two checkpoints pass without a page being written. Situations such as these do not indicate a buffer shortage.

- **Data management threshold (DMTH)—95%**

  This threshold is checked before a page is read or updated. If the threshold has not been exceeded, DB2 accesses the page in the virtual buffer pool once for each *page*, no matter how many rows are retrieved or updated in that page. If the threshold has been exceeded, DB2 accesses the page in the virtual buffer pool once for each *row* that is retrieved or updated in that page. In other words, retrieving or updating several rows in one page causes several page access operations.

Avoid reaching this threshold, because it has a significant effect on processor usage.

The DMTH is maintained for each individual virtual buffer pool. When the DMTH is reached in one virtual buffer pool, DB2 does *not* release pages from other virtual buffer pools.

- **Sequential prefetch threshold (SPTH)—90%**

  This threshold is checked at two different times:

  – Before scheduling a prefetch operation. If the threshold has been exceeded, the prefetch is not scheduled.

  – During buffer allocation for an already-scheduled prefetch operation. If the threshold has been exceeded, the prefetch is canceled.

  When the sequential prefetch threshold is reached, sequential prefetch is inhibited until more buffers become available. Operations that use sequential prefetch, such as those using large and frequent scans, are adversely affected.

## Thresholds you can change

You can change some thresholds directly, by using the ALTER BUFFERPOOL command. Changing a threshold in one virtual buffer pool or hiperpool has no effect on any other virtual buffer pool or hiperpool.

The variable thresholds are (from highest to lowest default value):

- **Sequential steal threshold (VPSEQT)**

  This threshold is a percentage of the virtual buffer pool that might be occupied by sequentially accessed pages. These pages can be in any state: updated, in-use, or available. Hence, any page might or might not count toward exceeding any other buffer pool threshold.

  The default value for this threshold is 80%. You can change that to any value from 0% to 100% by using the VPSEQT option of the ALTER BUFFERPOOL command.

  This threshold is checked before stealing a buffer for a sequentially accessed page instead of accessing the page in the virtual buffer pool. If the threshold has been exceeded, DB2 tries to steal a buffer holding a sequentially accessed page rather than one holding a randomly accessed page.

  Setting the threshold to 0% would prevent any sequential pages from taking up space in the virtual buffer pool. In this case, prefetch is disabled, and any sequentially accessed pages are discarded as soon as they are released.

  If you set VPSEQT to 0%, the value of HPSEQT is essentially meaningless: because when sequential pages are not kept in the virtual buffer pool, they have no chance of ever going to the hiperpool. But there is no restriction against having a non-zero value for HPSEQT with a zero value for VPSEQT.

  Setting the threshold to 100% allows sequential pages to monopolize the entire virtual buffer pool.

- **Hiperpool sequential steal threshold (HPSEQT)**

  This threshold is a percentage of the hiperpool that might be occupied by sequentially accessed pages. The effect of this threshold on the hiperpool is essentially the same as that of the sequential steal threshold on the virtual pool.

The default value for this threshold is 80%. You can change that to any value from 0% to 100% by using the HPSEQT option of the ALTER BUFFERPOOL command.

Because changed pages are not written to the hiperpool, HPSEQT is the only threshold for hiperpools.

- **Virtual buffer pool parallel sequential threshold (VPPSEQT)**

  This threshold is a portion of the virtual buffer pool that might be used to support parallel operations. It is measured as a percentage of the sequential steal threshold (VPSEQT). Setting VPPSEQT to zero disables parallel operation.

  The default value for this threshold is 50% of the sequential steal threshold (VPSEQT). You can change that to any value from 0% to 100% by using the VPPSEQT option on the ALTER BUFFERPOOL command.

- **Virtual buffer pool assisting parallel sequential threshold (VPXPSEQT)**

  This threshold is a portion of the virtual buffer pool that might be used to assist with parallel operations initiated from another DB2 in the data sharing group. It is measured as a percentage of VPPSEQT. Setting VPXPSEQT to zero disallows this DB2 from assisting with Sysplex query parallelism at run time for queries that use this buffer pool. For more information about Sysplex query parallelism, see Chapter 6 of *DB2 Data Sharing: Planning and Administration*.

  The default value for this threshold is 0% of the parallel sequential threshold (VPPSEQT). You can change that to any value from 0% to 100% by using the VPXPSEQT option on the ALTER BUFFERPOOL command.

- **Deferred write threshold (DWQT)**

  This threshold is a percentage of the virtual buffer pool that might be occupied by unavailable pages, including both updated pages and pages in use.

  The default value for this threshold is 50%. You can change that to any value from 0% to 90% by using the DWQT option on the ALTER BUFFERPOOL command.

  DB2 checks this threshold when an update to a page is completed. If the percentage of unavailable pages in the virtual buffer pool exceeds the threshold, write operations are scheduled for enough data sets (at up to 128 pages per data set) to decrease the number of unavailable buffers to 10% below the threshold. For example, if the threshold is 50%, the number of unavailable buffers is reduced to 40%.

  When the deferred write threshold is reached, the data sets with the oldest updated pages are written asynchronously. DB2 continues writing pages until the ratio goes below the threshold.

  ***Setting DWQT to 0:*** If you set DQWT to zero, then, to avoid synchronous writes to disk, DB2 implicitly uses the minimum value of (1% of the buffer pool, a specific number of pages). The number of pages is determined by the buffer pool page size, as shown in Table 75:

*Table 75. Number of change pages based on buffer pool size*

| Buffer pool page size | Number of changed pages |
|---|---|
| 4 KB | 40 |
| 8 KB | 24 |
| 16 KB | 16 |
| 32 KB | 12 |

- **Vertical deferred write threshold (VDWQT)**

  This threshold is similar to the deferred write threshold, but it applies to the number of updated pages for a single page set in the buffer pool. If the percentage or number of updated pages for the data set exceeds the threshold, writes are scheduled for that data set, up to 128 pages.

  You can specify this threshold in one of two ways:

  – As a *percentage* of the virtual buffer pool that might be occupied by updated pages from a single page set.

    The default value for this threshold is 10%. You can change the percentage to any value from 0% to 90%.

  – As the total *number of buffers* in the virtual buffer pool that might be occupied by updated pages from a single page set.

    You can specify the number of buffers from 0 to 9999. If you want to use the number of buffers as your threshold, you must set the percentage threshold to 0.

  ***Changing the threshold:*** Change the percent or number of buffers by using the VDWQT keyword on the ALTER BUFFERPOOL command.

  Because any buffers that count toward VDWQT also count toward DWQT, setting the VDWQT percentage higher than DWQT has no effect: DWQT is reached first, write operations are scheduled, and VDWQT is never reached. Therefore, the ALTER BUFFERPOOL command does not allow you to set the VDWQT percentage to a value greater than DWQT. You can specify a number of buffers for VDWQT than is higher than DWQT, but again, with no effect.

  This threshold is overridden by certain DB2 utilities, which use a constant limit of 64 pages rather than a percentage of the virtual buffer pool size. LOAD, REORG, and RECOVER use a constant limit of 128 pages.

  ***Setting VDWQT to 0:*** If you set VDQWT to zero for both the percentage and number of buffers, the minimum number of pages written is the same as for DWQT, shown in Table 75 on page 558.

## Guidelines for setting buffer pool thresholds

How you set buffer pools depends on your workload and the type and size of data being cached. But always think about the entire system when making buffer pool tuning decisions. See “Storage controller cache” on page 612 for more information about storage controller cache and the effect on DB2 performance.

For help in tuning your buffer pools, try the DB2 Bufferpool Tool feature of DB2.

***Pages are frequently re-referenced and updated:*** Suppose that you have a workload such as a branch table in a bank that contains a few hundred rows and is updated by every transaction. For such a workload, you want a high value for the deferred write and vertical deferred write threshold (90%). The result is that I/O is deferred until DB2 checkpoint and you have a lower I/O rate to disk.

However, if the set of pages updated exceeds the size of the virtual buffer pool, setting both DWQT and VDWQT to 90% might cause the sequential prefetch threshold (and possibly the data management threshold and the immediate write threshold) to be reached frequently. You might need to set DWQT and VDWQT lower in that case.

***Pages are rarely referenced:*** Suppose that you have a customer table in a bank that has millions of rows that are accessed randomly or are updated sequentially in batch. In this case, lowering the DWQT or VDWQT thresholds (perhaps down to 0) can avoid a surge of write I/Os caused by DB2 checkpoint. Lowering those thresholds causes the write I/Os to be distributed more evenly over time. Secondly, this can improve performance for the storage controller cache by avoiding the problem of flooding the device at DB2 checkpoint.

***Query-only buffer pools:*** For a buffer pool used exclusively for query processing, it is reasonable to set VPSEQT and HPSEQT to 100%. If parallel query processing is a large part of the workload, set VPPSEQT and, if applicable, VPXPSEQT, to a very high value.

***Mixed workloads:*** For a buffer pool used for both query and transaction processing, the values you set for VPSEQT and HPSEQT should depend on the respective priority of the two types of processing. The higher you set VPSEQT and HPSEQT, the better queries tend to perform, at the expense of transactions.

***Buffer pools containing LOBs:*** Put LOB data in buffer pools that are not shared with other data. For both LOG YES and LOG NO LOBs, use a deferred write threshold (DWQT) of 0. LOBs specified with LOG NO have their changed pages written at commit time (*force-at-commit* processing). If you set DWQT to 0, those writes happen continuously in the background rather than in a large surge at commit.

LOBs defined with LOG YES can use deferred write, but by setting DWQT to 0, you can avoid massive writes at DB2 checkpoints.

# Determining size and number of buffer pools

## Virtual buffer pool and hiperpool sizes

Initially, you set the sizes (in number of pages) of your virtual buffer pools and hiperpools on installation panels DSNTIP1, DSNTIP2, and DSNTIP6. Because you can modify the sizes of virtual buffer pools and hiperpools using the ALTER BUFFERPOOL command, it is not important to choose an exact size initially.

## The buffer pool hit ratio

Considering the real storage and expanded storage that is available to DB2, you can help some of your applications and queries by making the virtual buffer pools large enough to increase the *buffer hit ratio*. Buffer hit ratio is a measure of how often a page access (a getpage) is satisfied without requiring an I/O operation.

Accounting reports, which are application related, show the hit ratio for specific applications. An accounting trace report shows the ratio for single threads. The DB2 PM buffer pool statistics report shows the hit ratio for the subsystem as a whole. For example, the buffer pool hit ratio is shown in field **A** in Figure 64 on page 568. The buffer hit ratio uses the following formula to determine how many getpage operations did not require an I/O operation:

```
Hit ratio = getpages - pages_read_from_DASD / getpages
```

where pages_read_from_DASD is the sum of the following fields:

- Number of synchronous reads (field **B** in Figure 64 on page 568)
- Number of pages read via sequential prefetch (field **C**)
- Number of pages read via list prefetch (field **D**)
- Number of pages read via dynamic prefetch (field **E**)

***Example:*** If you have 1000 getpages and 100 pages were read from DASD, the equation would be as follows:

```
Hit ratio = (1000-100)/1000
```

The hit ratio in this case is 0.9.

***Highest hit ratio:*** The highest possible value for the hit ratio is 1.0, which is achieved when every page requested is always in the buffer pool. Reading index non-leaf pages tend to have a very high hit ratio since they are frequently re-referenced and thus tend to stay in the buffer pool.

***Lowest hit ratio:*** The lowest hit ratio occurs when the requested page is not in the buffer pool; in this case, the hit ratio is 0 or less. A negative hit ratio means that prefetch has brought pages into the buffer pool that are not subsequently referenced. The pages are not referenced because either the query stops before it reaches the end of the table space or DB2 must take the pages away to make room for newer ones before the query can access them.

***A low hit ratio is not always bad:*** While it might seem desirable to make the buffer hit ratio as close to 1.0 as possible, do not automatically assume a low buffer pool hit ratio is bad. The hit ratio is a relative value, based on the type of application. For example, an application that browses huge amounts of data using table space scans might very well have a buffer pool hit ratio of 0. What you want to watch for is those cases where the hit ratio drops significantly for the same application. In those cases, it might be helpful to investigate further.

***Hit ratios for additional processes:*** The hit ratio measurement becomes less meaningful if the buffer pool is being used by additional processes, such as work files or utilities. Some utilities and SQL statements use a special type of getpage request that reserve an empty buffer without requiring that the page be read from disk.

A getpage is issued for each empty work file page without read I/O during sort input processing. The hit ratio can be calculated if the work files are isolated in their own buffer pools. If they are, then the number of getpages used for the hit ratio formula is divided in half as follows:

```
Hit ratio = ((getpages / 2) - pages_read_from_DASD) / (getpages / 2)
```

## Buffer pool size guidelines

DB2 handles large virtual buffer pools very efficiently. Searching in large virtual buffer pools (100 MB or more) does not use any more of the processor’s resources than searching in smaller pools. There is a slight increase in processing overhead when buffer pools are in data spaces backed up by real storage. The increase is larger if the buffer pool is backed by a mixture of real and expanded storage. Never define buffer pools that are backed by auxiliary storage.

If virtual pool storage is not backed by real storage, the resulting paging activity is inefficient. If you see paging, increase the amount of real storage, or shrink down the virtual pools size and use hiperpools.

***Problems with paging:*** Paging occurs when the virtual storage size requirements for a buffer pool exceeds the real storage capacity for the OS/390 image. In this case, least recently used data pages in the buffer pool are migrated to expanded or auxiliary storage. Subsequent access to these pages results in a page fault and the page must be brought into real storage from auxiliary or expanded storage. Paging of buffer pool storage, especially that from auxiliary to real, can negatively impact

DB2 performance. The statistics for PAGE-INS REQUIRED FOR WRITE and PAGE-INS REQUIRED FOR READ shown in Figure 64 on page 568 are useful in determining if the buffer pool size setting is too large for available real storage.

If the large buffer pool size results in excessive real storage paging to expanded storage, consider using hiperpools.

### Advantages of large buffer pools

In general, larger buffer pool sizes can:

- Result in a higher buffer pool hit ratio, which can reduce the number of I/O operations. Fewer I/O operations can reduce I/O contention, which can provide better response time and reduce the processor resource needed for I/O operations.
- Give an opportunity to achieve higher transaction rates with the same response time. For any given response time, the transaction rate depends greatly on buffer pool size.
- Prevent I/O contention for the most frequently used disks, particularly the catalog tables and frequently referenced user tables and indexes. In addition, a large buffer pool is beneficial when a DB2 sort is used during a query, because I/O contention on the disks containing the work file table spaces is reduced.

### Choosing one or many buffer pools

Whether you choose one or many 4-KB buffer pools, make sure to have at least one 32- KB buffer pool that DB2 can use, even if all tables use 4 KB pages. Some SQL operations, such as joins, can create a result row that does not fit into a 4 KB page.

***Reasons to choose a single buffer pool:*** If your system has any or all of the following conditions, it is probably best to choose a single 4 KB buffer pool:

- It does not have enough total buffer space for more than 10000 4-KB buffers.
- You have no one with the application knowledge necessary to do more specialized tuning.
- It is a test system.

***Reasons to choose more than one buffer pool:*** The following are some advantages to having more than one buffer pool:

- You can isolate data in separate buffer pools to favor certain applications, data, and indexes.

  For example, if you have large buffer pools, putting indexes into separate pools from data might improve performance. You might want to put tables and indexes that are updated frequently into a buffer pool with different characteristics from those that are frequently accessed but infrequently updated.
- You can put work files into a separate buffer pool. This can provide better performance for sort-intensive queries. Applications that use created temporary tables use work files for those tables. Keeping work files separate allows you to monitor temporary table activity more easily.
- This process of segregating different activities and data into separate buffer pools has the advantage of providing good and relatively inexpensive performance diagnosis data from statistics and accounting traces.

## Choosing a page-stealing algorithm

When DB2 must take away a page in the buffer pool to make room for a newer page, this action is called “stealing” the page from the buffer pool. DB2 usually uses a least-recently-used (LRU) algorithm for managing pages in storage. That is, it takes away older pages so that more recently used pages can remain in the virtual buffer pool.

However, using the ALTER BUFFERPOOL command, you can also choose to have DB2 use a first-in, first-out (FIFO) algorithm. With this simple algorithm, DB2 does not keep track of how often a page is referenced—the pages that are oldest are moved out, no matter how frequently they are referenced. This simple approach to page stealing results in a small decrease in the cost of doing a getpage operation, and it can reduce internal DB2 latch contention in environments that require very high concurrency.

**Recommendations:**

- In most cases, keep the default, LRU.
- Use FIFO for buffer pools that have no I/O; that is, the table space or index remains in the buffer pool. Because all the pages are there, there is no need to pay the additional cost of a more complicated page management algorithm.
- Keep objects that can benefit from the FIFO algorithm in different buffer pools from those that benefit from the LRU algorithm. See options for PGSTEAL in ALTER BUFFERPOOL command in *DB2 Command Reference*.

## Monitoring and tuning buffer pools using online commands

The information under this heading, up to “Using DB2 PM to monitor buffer pool statistics” on page 567, is General-use Programming Interface and Associated Guidance Information as defined in “Notices” on page 1095.

The DISPLAY BUFFERPOOL and ALTER BUFFERPOOL commands allow you to monitor and tune buffer pools and hiperpools on line, while DB2 is running, without the overhead of running traces.

You can use the ALTER BUFFERPOOL command to change the following attributes:

- Size
- Thresholds
- Virtual pool type (primary or data space)
- Hiperpool castout attribute
- Page stealing algorithm

You can use the DISPLAY BUFFERPOOL command to display the current status of one or more active or inactive buffer pools. For example, the following command:

```
DISPLAY BUFFERPOOL(BP1) DETAIL
```

produces a detailed report of the status of BP1, as shown in Figure 62 on page 564. The operation captured by this report is the processing of sort work files for a query.

```
+DISPLAY BPOOL(BP1) DETAIL
DSNB401I + BUFFERPOOL NAME BP1, BUFFERPOOL ID 1, USE COUNT 8
DSNB402I + VIRTUAL BUFFERPOOL SIZE =  6000 BUFFERS
             ALLOCATED       =     6000   TO BE DELETED   =        0
             IN-USE/UPDATED  =       11
DSNB406I + VIRTUAL BUFFERPOOL TYPE -
             CURRENT           = PRIMARY
             PENDING           = PRIMARY
           PAGE STEALING METHOD = LRU
DSNB403I + HIPERPOOL SIZE = 0 BUFFERS, CASTOUT = YES
             ALLOCATED       =        0   TO BE DELETED   =        0
             BACKED BY ES    =        0
DSNB404I + THRESHOLDS -
             VP SEQUENTIAL         = 80   HP SEQUENTIAL          = 80
             DEFERRED WRITE        = 50   VERTICAL DEFERRED WRT  = 10, 0
             PARALLEL SEQUENTIAL   =  0   ASSISTING PARALLEL SEQT= 0
DSNB409I + INCREMENTAL STATISTICS SINCE 14:57:55 JAN 22, yyyy
DSNB411I + RANDOM GETPAGE    =      156 SYNC READ I/O (R) =        3
           SEQ.   GETPAGE    =   132294 SYNC READ I/O (S) =A    326
           DMTH HIT          =        0
DSNB412I + SEQUENTIAL PREFETCH -
                                                     C
             REQUESTS B      =     8253   PREFETCH I/O =     4461
             PAGES READ D    =    35660
DSNB413I + LIST PREFETCH -
             REQUESTS        =        0   PREFETCH I/O    =        0
             PAGES READ      =        0
DSNB414I + DYNAMIC PREFETCH -
             REQUESTS        =        0   PREFETCH I/O    =        0
             PAGES READ      =        0
DSNB415I + PREFETCH DISABLED -
             NO BUFFER       =        0   NO READ ENGINE  =        0
                                                          F
DSNB420I + SYS PAGE UPDATES  =E 137857 SYS PAGES WRITTEN = 63320
           ASYNC WRITE I/O   =     2057 SYNC WRITE I/O    =        0
DSNB421I + DWT HIT G         =       27 VERTICAL DWT HIT  H=    231
           NO WRITE ENGINE   =        0
DSNB430I + HIPERPOOL ACTIVITY (NOT USING ASYNCHRONOUS
             DATA MOVER FACILITY) -
             SYNC HP READS   =        0   SYNC HP WRITES  =        0
             ASYNC HP READS  =        0   ASYNC HP WRITES =        0
             READ FAILURES   =        0   WRITE FAILURES  =        0
DSNB431I + HIPERPOOL ACTIVITY (USING ASYNCHRONOUS
             DATA MOVER FACILITY) -
             HP READS        =        0   HP WRITES       =        0
             READ FAILURES   =        0   WRITE FAILURES  =        0
DSNB440I + PARALLEL ACTIVITY -
            PARALLEL REQUEST  =       0  DEGRADED PARALLEL=        0

DSN9022I + DSNB1CMD '+DISPLAY BPOOL' NORMAL COMPLETION
```

*Figure 62. Sample output from the DISPLAY BUFFERPOOL command. This sample output shows buffer pool statistics for the processing of sort work files.*

In Figure 62, find the following fields:

- SYNC READ I/O (S) ( A ) shows the number of sequential synchronous read I/O operations. Sequential synchronous read I/Os occur when prefetch is disabled or when the requested pages are not consecutive. One way to decrease the value of 326, which might be high for this application, is to increase the buffer pool size until the number of read I/Os decreases while avoiding paging.

  To determine the total number of synchronous read I/Os, add SYNC READ I/O (S) and SYNC READ I/O (R).

- In message DSNB412I, REQUESTS ( B ) shows the number of times that sequential prefetch was triggered, and PREFETCH I/O ( C ) shows the number of times that sequential prefetch occurred. PAGES READ ( D ) shows the number of pages read using sequential prefetch. If you divide the PAGES READ value by the PREFETCH I/O, you get 7.99. This is because the prefetch quantity for sort work files is 8 pages. For operations other than sorts, the prefetch quantity could be up to 32 pages, depending on the application.
- SYS PAGE UPDATES ( E ) corresponds to the number of buffer updates.
- SYS PAGES WRITTEN ( F ) is the number of pages written to disk.
- DWT HIT ( G ) is the number of times the deferred write threshold (DWQT) was reached. This number is workload dependent.
- VERTICAL DWT HIT ( H ) is the number of times the vertical deferred write threshold (VDWQT) was reached. This value is per data set, and it is related to the number of asynchronous writes.

Because the number of synchronous read I/Os ( A ) and the number of sequential prefetch I/Os ( C ) are relatively high, you would want to tune the buffer pools by changing the buffer pool specifications. For example, you could make the buffer operations more efficient by defining a hiperpool if you have expanded storage on your machine. To do that, enter the following command:

```
-ALTER BUFFERPOOL(BP1) VPSIZE(6000) HPSIZE(20000) CASTOUT(NO)
```

After issuing the previous ALTER BUFFERPOOL command, you can see the resulting changes in the virtual buffer pool and hiperpool by issuing the DISPLAY BUFFERPOOL command again. The output is shown in Figure 63 on page 566.

```
+DISPLAY BPOOL(BP1) DETAIL
DSNB401I + BUFFERPOOL NAME BP1, BUFFERPOOL ID 1, USE COUNT 8
DSNB402I + VIRTUAL BUFFERPOOL SIZE =  6000 BUFFERS
             ALLOCATED       =     6000   TO BE DELETED   =        0
             IN-USE/UPDATED  =       11
DSNB406I + VIRTUAL BUFFERPOOL TYPE -
             CURRENT           = PRIMARY
             PENDING           = PRIMARY
           PAGE STEALING METHOD = LRU
DSNB403I + HIPERPOOL SIZE I  = 20000 BUFFERS, CASTOUT = NO
             ALLOCATED J     =    20000   TO BE DELETED   =        0
             BACKED BY ES K  =    13929
DSNB404I + THRESHOLDS -
             VP SEQUENTIAL         = 80   HP SEQUENTIAL          = 80
             DEFERRED WRITE        = 50   VERTICAL DEFERRED WRT  = 10,0
             PARALLEL SEQUENTIAL   =  0   ASSISTING PARALLEL SEQT= 0
DSNB405I + HIPERSPACE NAME(S) - @011D31A
DSNB409I + INCREMENTAL STATISTICS SINCE 16:16:16 JAN 23, yyyy
DSNB411I + RANDOM GETPAGE    =      156 SYNC READ I/O (R) =       11
           SEQ.   GETPAGE    =   132294 SYNC READ I/O (S) = L      0
           DMTH HIT          =        0
DSNB412I + SEQUENTIAL PREFETCH -
             REQUESTS        =     8253   PREFETCH I/O M  =      103
             PAGES READ N    =      633
DSNB413I + LIST PREFETCH -
             REQUESTS        =        0   PREFETCH I/O    =        0
             PAGES READ      =        0
DSNB414I + DYNAMIC PREFETCH -
             REQUESTS        =        0   PREFETCH I/O    =        0
             PAGES READ      =        0
DSNB415I + PREFETCH DISABLED -
             NO BUFFER       =        0   NO READ ENGINE  =        0
DSNB420I + SYS PAGE UPDATES  =   137857 SYS PAGES WRITTEN =    63338
           ASYNC WRITE I/O   =     2141 SYNC WRITE I/O    =        2
DSNB421I + DWT HIT           =      135 VERTICAL DWT HIT  =      226
           NO WRITE ENGINE   =        2
DSNB430I + HIPERPOOL ACTIVITY (NOT USING ASYNCHRONOUS
             DATA MOVER FACILITY) -
             SYNC HP READS O =      327   SYNC HP WRITES  =        0
             ASYNC HP READS  =        0   ASYNC HP WRITES =        0
             READ FAILURES   =        0   WRITE FAILURES  =        0
DSNB431I + HIPERPOOL ACTIVITY (USING ASYNCHRONOUS
             DATA MOVER FACILITY) -
                                                 Q
             HP READS P      =    35177   HP WRITES       =    35657
             READ FAILURES   =        0   WRITE FAILURES  = R      0
DSNB440I + PARALLEL ACTIVITY -
            PARALLEL REQUEST =        0  DEGRADED PARALLEL=        0

DSN9022I + DSNB1CMD '+DISPLAY BPOOL' NORMAL COMPLETION
```

*Figure 63. Sample output from the DISPLAY BUFFERPOOL command. This output shows how the buffer pool statistics changed after the ALTER BUFFERPOOL command was issued.*

In Figure 63, notice the following fields:

- You can verify the new hiperpool size by checking the HIPERPOOL SIZE field ( **I** ).
- In this example, the hiperpool size allocated (ALLOCATED **J** ) is larger than the value for BACKED BY ES ( **K** ) because the hiperpool was larger than necessary. The value for ALLOCATED can also be larger than the BACKED BY ES value when there is not enough expanded storage available to support the hiperpool size you specified. If the available expanded storage had been exceeded, there would be a non-zero value in the WRITE FAILURES field ( **R** ).

- The value for SYNC READ I/O ( L ), which was 326 before the ALTER BUFFERPOOL command was issued, has decreased significantly.
- The values for PREFETCH I/O ( M ) and PAGES READ( N ) have decreased significantly because most of the requested pages are in the hiperpool, resulting in fewer pages that need to be fetched from disk through sequential prefetch.
- SYNC HP READS ( O ) corresponds to the SYNC READ I/O (S) ( A ) value in Figure 62 on page 564.
- HP READS ( P ) shows the number of times data was read from the hiperpool into the virtual buffer pool.
- HP WRITES ( Q ) shows the number of times data was written to the hiperpool from the virtual buffer pool.

To obtain buffer pool information on a specific data set, you can use the LSTATS option of the DISPLAY BUFFERPOOL command. For example, you can use the LSTATS option to:

- Provide page count statistics for a certain index. With this information, you could determine whether a query used the index in question, and perhaps drop the index if it was not used.
- Monitor the response times on a particular data set. If you determine that I/O contention is occurring, you could redistribute the data sets across your available disks.

This same information is available with IFCID 0199 (statistics class 8).

For more information on the ALTER BUFFERPOOL or DISPLAY BUFFERPOOL commands, see Chapter 2 of *DB2 Command Reference*.

## Using DB2 PM to monitor buffer pool statistics

You can find information about the database buffer pools in the statistics report produced by DB2 PM, as Figure 64 on page 568 shows.

```
TOT4K  READ OPERATIONS       QUANTITY   TOT4K  WRITE OPERATIONS      QUANTITY
---------------------------  --------   ---------------------------  --------
BPOOL HIT RATIO (%) A           73.12   BUFFER UPDATES                 220.4K
                                        PAGES WRITTEN                35169.00
GETPAGE REQUEST               1869.7K   BUFF.UPDATES/PAGES WRITTEN H     6.27
GETPAGE REQUEST-SEQUENTIAL    1378.5K
GETPAGE REQUEST-RANDOM         491.2K   SYNCHRONOUS WRITES I
                                        ASYNCHRONOUS WRITES           5084.00
SYNCHRONOUS READS B          54187.00
SYNCHRON. READS-SEQUENTIAL   35994.00   PAGES WRITTEN PER WRITE I/O J    5.78
SYNCHRON. READS-RANDOM       18193.00
                                        HORIZ.DEF.WRITE THRESHOLD        2.00
GETPAGE PER SYN.READ-RANDOM     27.00   VERTI.DEF.WRITE THRESHOLD        0.00
                                        DM THRESHOLD K          0.00
SEQUENTIAL PREFETCH REQUEST  41800.00   WRITE ENGINE NOT AVAILABLE L     0.00
SEQUENTIAL PREFETCH READS    14473.00
PAGES READ VIA SEQ.PREFETCH C  444.0K   SYNC.HPOOL WRITE                 0.00
S.PRF.PAGES READ/S.PRF.READ     30.68   ASYNC.HPOOL WRITE             5967.00
                                        HPOOL WRITE FAILED               0.00
LIST PREFETCH REQUESTS        9046.00   ASYN.DA.MOVER HPOOL WRITE-S    523.2K
LIST PREFETCH READS           2263.00   ASYN.DA.MOVER HPOOL WRITE-F      0.00
PAGES READ VIA LST PREFETCH D 3046.00
L.PRF.PAGES READ/L.PRF.READ      1.35   PAGE-INS REQUIRED FOR WRITE     45.00

DYNAMIC PREFETCH REQUESTED    6680.00
DYNAMIC PREFETCH READS         142.00
PAGES READ VIA DYN.PREFETCH E 1333.00
D.PRF.PAGES READ/D.PRF.READ      9.39

PREF.DISABLED-NO BUFFER F        0.00
PREF.DISABLED-NO READ ENG G      0.00

SYNC.HPOOL READ               7194.00
ASYNC.HPOOL READ              1278.00
HPOOL READ FAILED                0.00
ASYN.DA.MOVER HPOOL READ-S   58983.00
ASYN.DA.MOVER HPOOL READ-F       0.00

PAGE-INS REQUIRED FOR READ     460.4K
```

*Figure 64. DB2 PM database buffer pool statistics (modified)*

The formula for the buffer pool hit ratio (fields A through E) is explained in “The buffer pool hit ratio” on page 560

Increase the virtual buffer pool size or reduce the workload if:

- Sequential prefetch is inhibited. PREF.DISABLED-NO BUFFER (F) shows how many times sequential prefetch is disabled because the sequential prefetch threshold (90% of the pages in the buffer pool are unavailable) has been reached.
- You detect poor update efficiency. You can determine update efficiency by checking the values in both of the following fields:
  - – BUFF.UPDATES/PAGES WRITTEN (H)
  - – PAGES WRITTEN PER WRITE I/O (J)

  In evaluating the values you see in these fields, keep in mind that there are no absolute acceptable or unacceptable values. Each installation’s workload is a special case. To assess the update efficiency of your system, monitor for overall trends rather than for absolute high values for these ratios.

The following factors impact buffer updates per pages written and pages written per write I/O:

- Sequential nature of updates
- Number of rows per page
- Row update frequency

For example, a batch program that processes a table in skip sequential mode with a high row update frequency in a dedicated environment can achieve very good update efficiency. In contrast, update efficiency tends to be lower for transaction processing applications, because transaction processing tends to be random.

The following factors affect the ratio of pages written per write I/O:

- *Checkpoint frequency.* The CHECKPOINT FREQ field on panel DSNTIPN specifies the number of consecutive log records written between DB2 system checkpoints. At checkpoint time, I/Os are scheduled to write all updated pages on the deferred write queue to disk. If system checkpoints occur too frequently, the deferred write queue does not grow large enough to achieve a high ratio of pages written per write I/O.
- *Frequency of active log switch.* DB2 takes a system checkpoint each time the active log is switched. If the active log data sets are too small, checkpoints occur often, which prevents the deferred write queue from growing large enough to achieve a high ratio of pages written per write I/O. For recommendations on active log data set size, see “Log capacity” on page 602.
- *Buffer pool size.* The deferred write thresholds (VDWQT and DWQT) are a function of buffer pool size. If the buffer pool size is decreased, these thresholds are reached more frequently, causing I/Os to be scheduled more often to write some of the pages on the deferred write queue to disk. This prevents the deferred write queue from growing large enough to achieve a high ratio of pages written per write I/O.
- *Number of data sets, and the spread of updated pages across them.* The maximum number of pages written per write I/O is 32, subject to a limiting scope of 150 pages (roughly one cylinder). For example, if your application updates page 2 and page 149 in a series of pages, the two changed pages could potentially be written with one write I/O. But if your application updates page 2 and page 155 within a series of pages, writing the two changed pages would require two write I/Os because of the 150-page limit. Updated pages are placed in a deferred write queue based on the data set. For batch processing it is possible to achieve a high ratio of pages written per write I/O, but for transaction processing the ratio is typically lower.

  For LOAD, REORG, and RECOVER, the maximum number of pages written per write I/O is 64, and there is no limiting scope.

- SYNCHRONOUS WRITES ( I ) is a high value. This field counts the number of immediate writes. However, immediate writes are not the only type of synchronous write; thus, it is difficult to provide a monitoring value for the number of immediate writes.

  Ignore SYNCHRONOUS WRITES when DM THRESHOLD is zero.

- DM THRESHOLD ( K ) is reached. This field shows how many times a page was immediately released because the data management threshold was reached. The quantity listed for this field should be zero.

Also note the following fields:

- WRITE ENGINE NOT AVAILABLE ( L )

This field records the number of times that asynchronous writes were deferred because DB2 reached its maximum number of concurrent writes. You cannot change this maximum value. This field has a nonzero value occasionally.

- PREF.DISABLED-NO READ ENG ( G )

  This field records the number of times that a sequential prefetch was not performed because the maximum number of concurrent sequential prefetches was reached. Instead, normal reads were done. You cannot change this maximum value.

## Tuning the EDM pool

During the installation process, DSNTINST CLIST calculates the size of the EDM pool. The EDM pool contains:

- Database descriptors (DBDs)
- Skeleton cursor tables (SKCTs)
- Cursor tables (CTs), or copies of the SKCTs
- Skeleton package tables (SKPTs)
- Package tables (PTs), or copies of the SKPTs
- An authorization cache block for each plan, excluding plans that you created specifying CACHESIZE(0)
- Skeletons of dynamic SQL if your installation has YES for the CACHE DYNAMIC SQL field of installation panel DSNTIP4

If you have YES for the CACHE DYNAMIC SQL and have a size specified for the DATASPACE, the skeletons are kept in the DATASPACE and some internal structures are kept in the EDM pool. See “Allied thread allocation” on page 620 for information on how SKCTs, CTs, and DBDs are handled.

You can check the calculated size of the EDM pool on panel DSNTIPC. Refer to *DB2 Installation Guide* for more information on specifying the size of the EDM pool.

For data sharing, you might need to increase the EDM pool storage estimate. For more information, see Chapter 2 of *DB2 Data Sharing: Planning and Administration*.

Because of an internal process that changes the size of plans initially bound in one release and then are rebound in a later release, you should carefully monitor the size of the EDM pool and increase its size, if necessary. For information about how to estimate the size of the EDM pool, see *DB2 Installation Guide*.

## EDM pool space handling

When pages are needed for the EDM pool, any pages that are available are allocated first. If the available pages do not provide enough space to satisfy the request, pages are “stolen” from an inactive SKCT, SKPT, DBD, or dynamic SQL skeleton. If there is still not enough space, an SQL error code is sent to the application program.

You should design the EDM pool to contain:

- The CTs, PTs, and DBDs in use
- The SKCTs for the most frequently used applications
- The SKPTs for the most frequently used applications
- The DBDs referred to by these applications
- The cache blocks for your plans that have caches

- The skeletons of the most frequently used dynamic SQL statements, if your system has enabled the dynamic statement cache

By designing the EDM pool this way, you can avoid allocation I/Os, which can represent a significant part of the total number of I/Os for a transaction. You can also reduce the processing time necessary to check whether users attempting to execute a plan are authorized to do so.

An EDM pool that is too small causes:

- Increased I/O activity in DSNDB01.SCT02, DSNDB01.SPT01, and DSNDB01.DBD01.
- Increased response times, due to loading the SKCTs, SKPTs, and DBDs. If caching of dynamic SQL is used and the needed SQL statement is not in the EDM pool, that statement has to be prepared again.
- Fewer threads used concurrently, due to a lack of storage.

## Implications for database design

When you design your databases, be aware that a very large number of objects in your database means a larger DBD for that database. And when you drop objects, storage is not automatically reclaimed in that DBD, which can mean that DB2 must take more locks for the DBD. To reclaim storage in the DBD, use the MODIFY utility, as described in Part 2 of *DB2 Utility Guide and Reference*.

Large DBDs can cause problems when not enough contiguous space is available to load them into the EDM pool. You can store large DBDs in 32K pieces to avoid loading problems. Large DBDs created in a release of DB2 prior to Version 6 will not be stored in 32K pieces until a DDL causes the DBD to be written out.

## Monitoring and tuning the EDM pool

The DB2 statistics record provides information on the EDM pool. Figure 65 on page 572 shows how DB2 PM presents this information in the statistics report.

```
EDM POOL                      QUANTITY
---------------------------   --------
PAGES IN EDM POOL  A           2500.00
  HELD BY DBDS                  245.00
  HELD BY CTS                    24.00
  HELD BY SKCTS                  12.00
  HELD BY SKPTS                   0.00
  HELD BY PTS                     0.00
  FREE PAGES       B           1917.96
% PAGES IN USE                   11.64
% NON STEAL. PAGES IN USE         0.14

FAILS DUE TO POOL FULL            0.00

DBD REQUESTS                    135.18
DBD NOT IN EDM POOL               0.00
DBD HIT RATIO (%)  C               N/C
CT REQUESTS                       1.42
CT NOT IN EDM POOL                0.00
CT HIT RATIO (%)   D               N/C
PT REQUESTS                       0.00
PT NOT IN EDM POOL                0.00
PT HIT RATIO (%)   E               N/C

PAGES FOR DYN SQL CACHE          10.82
PAGES IN DATASPACE                0.00
FREE PAGES IN DATASPACE           0.00
FAILS DUE TO DATASPACE FULL       0.00

DYNAMIC SQL STMT              QUANTITY
---------------------------   --------
PREPARE REQUESTS   F           4912.42
  FULL PREPARES    G             10.89
  SHORT PREPARES                  0.00
GLOBAL CACHE HIT RATIO (%)  H     1.00

IMPLICIT PREPARES                 0.00
STMT INVALID (MAXKEEPD)           0.00
STMT INVALID (DDL)                0.00
LOCAL CACHE HIT RATIO (%)         1.00
```

*Figure 65. EDM pool utilization in the DB2 PM statistics report*

The important values to monitor are:

***Efficiency of the pool:*** You can measure the efficiency of the EDM pool by using the following ratios:

DBD HIT RATIO (%) C
CT HIT RATIO (%) D
PT HIT RATIO (%) E

These ratios for the EDM pool depend upon your location's work load. In most DB2 subsystems, a value of 5 or more is acceptable. This means that at least 80% of the requests were satisfied without I/O.

The number of free pages is shown in FREE PAGES ( B ) in Figure 65. If this value is more than 20% of PAGES IN EDM POOL ( A ) during peak periods, the EDM pool size is probably too large. In this case, you can reduce its size without affecting the efficiency ratios significantly.

***EDM pool hit ratio for cached dynamic SQL:*** If you have caching turned on for dynamic SQL, the EDM pool statistics have information that can help you determine how successful your applications are at finding statements in the cache. See mapping macro DSNDQISE for descriptions of these fields.

PREPARE REQUESTS ( **F** ) in Figure 65 records the number of requests to search the cache. FULL PREPARES ( **G** ) records the number of times that a statement was inserted into the cache, which can be interpreted as the number of times a statement was not found in the cache. To determine how often the dynamic statement was used from the cache, check the value in GLOBAL CACHE HIT RATIO ( **H** ). The value is calculated with the following formula:

```
(PREPARE REQUESTS – FULL PREPARES) ⁄ PREPARE REQUESTS = hit ratio
```

***EDM pool space utilization and performance:*** For smaller EDM pools, space utilization or fragmentation is normally more critical than for larger EDM pools. For larger EDM pools, performance is normally more critical. DB2 emphasizes performance and uses less optimum EDM storage allocation when the EDM pool size exceeds 40 megabytes. For systems with large EDM pools that are greater than 40 megabytes to continue to use optimum EDM storage allocation at the cost of performance, you can set the keyword EDMBFIT in the DSNTIJUZ job to YES. The EDMBFIT keyword adjusts the search algorithm on systems with EDM pools that are larger than 40 megabytes. The default NO tells DB2 to use a first-fit algorithm while YES tells DB2 to use a better-fit algorithm. YES is a better choice when EDMPOOL full conditions occur for even a very large EDM pool or the number of current threads is not very high for an EDM pool size that exceeds 40 megabytes.

## Tips for managing EDM pool storage

Here are guidelines for helping keep EDM pool storage under control.

### Use packages

By using multiple packages you can increase the effectiveness of EDM pool storage management by having smaller objects in the pool.

### Use RELEASE(COMMIT) when appropriate

Using the bind option RELEASE(COMMIT) for infrequently used packages and plans can cause objects to be removed from the EDM pool sooner.

### Release thread storage

If your EDM pool storage grows continually, consider having DB2 periodically free unused thread storage. To do this, specify YES for CONTRACT THREAD STG on installation panel DSNTIPE. This option can affect performance and is best used when your system has many long-running threads and your EDM storage is constrained.

### Understand the impact of using DEGREE(ANY)

A plan or package that is bound with DEGREE(ANY) can require 50 to 70% more storage for the CTs and PTs in the EDM pool than one bound with DEGREE(1). If you change a plan or package to DEGREE(ANY), check the change in the column AVGSIZE in SYSPLAN or SYSPACKAGE to determine the increase required.

### Put dynamic statement cache in a data space

When you use dynamic statement caching, the EDM storage that is used to contain cached dynamic statements is moved to a data space by default. (Field EDMPOOL DATA SPACE SIZE on installation panel DSNTIPC specifies the amount of storage that is moved; if a value is specified in this field, DB2 uses a calculated value.) However, as explained under “Advantages of data spaces” on page 553, the use of a data space is recommended only if you are constrained on storage in *ssnm*DBM1 or have a processor with an operating system that supports more than 2 GB of real storage. To prevent a data space from being used, set field EDMPOOL DATA SPACE SIZE to zero. If the use of a data space is appropriate and you want to change the amount of EDM storage that is moved there, set field EDMPOOL DATA SPACE SIZE to the desired value.

## Increasing RID pool size

The RID pool is used for all record identifier (RID) processing. It is used for enforcing unique keys while updating multiple rows. and for sorting RIDs during the following operations:

List prefetch, including single index list prefetch

Access via multiple indexes

Hybrid joins

To favor the selection and efficient completion of those access paths, increase the maximum RID pool size. However, if there is not enough RID pool storage, it is possible that the statement might revert to a table space scan.

A RID pool size of 0 disables those access paths. If you specify a RID pool size of 0, plans or packages that were previously bound with a non-zero RID pool size might experience significant performance degradation. Rebind any plans or packages that include SQL statements that use RID processing.

The default RID pool size is 4 MB. You can override this value on panel DSNTIPC.

To determine if a transaction used the RID pool, see the RID Pool Processing section of the DB2 PM accounting trace record.

The RID pool, which all concurrent work shares, is limited to a maximum of 1000 MB. The RID pool is created at system initialization, but no space is allocated until RID storage is needed. It is then allocated above the 16 MB line in 16 KB blocks as needed, until the maximum size you specified on installation panel DSNTIPC is reached.

The general formula for computing RID pool size is:

```
Number of concurrent RID processing activities ×
  average number of RIDs × 2 × 5 bytes per RID
```

For example, three concurrent RID processing activities, with an average of 4000 RIDs each, would require 120 KB of storage, because:

```
3 × 4000 × 2 × 5 = 120KB
```

Whether your SQL statements that use RID processing complete efficiently or not depends on other concurrent work using the RID pool.

## Controlling sort pool size and sort processing

Sort is invoked when a cursor is opened for a SELECT statement that requires sorting. The maximum size of the sort work area allocated for each concurrent sort user depends on the value you specified for the SORT POOL SIZE field on installation panel DSNTIPC. The default value is 1 MB.

## Estimating the maximum size of the sort pool

You can change the maximum size of the sort pool by using the installation panels in UPDATE mode. A rough formula for determining the maximum size is as follows:

```
16000 × (12 + sort key length + sort data length + 4 (if hardware sort))
```

For sort key length and sort data length, use values that represent the maximum values for the queries you run. To determine these values, refer to fields QW0096KL (key length) and QW0096DL (data length) in IFCID 0096, as mapped by macro DSNDQW01. You can also determine these values from an SQL activity trace.

If a column is in the ORDER BY clause that is not in the select clause, that column should be included in the sort data length and the sort key length as shown in the following example:

```
SELECT C1, C2, C3
 FROM tablex
 ORDER BY C1, C4;
```

If C1, C2, C3, and C4 are each 10 bytes in length for an MVS/ESA system, you could estimate the sort pool size as follows:

```
16000 × (12 + 4 + 20 + (10 + 10 + 10 + 10)) =1216000 bytes

where:  16000 = maximum number of sort nodes
           12 = size (in bytes) of each node
            4 = number of bytes added for each node if
                sort facility hardware used
           20 = sort key length (ORDER BY C1, C4)
  10+10+10+10 = sort data length (each column is 10 bytes in length)
```

## Understanding how sort work files are allocated

The sort begins with the input phase when ordered sets of rows are written to work files. At the end of the input phase, when all the rows have been sorted and inserted into the work files, the work files are merged together, if necessary, into one work file containing the sorted data. The merge phase is skipped if there is only one work file at the end of the input phase. In some cases, intermediate merging might be needed if the maximum number of sort work files has been allocated.

The work files used in sort are logical work files, which reside in work file table spaces in your work file database (which is DSNDB07 in a non data-sharing environment). DB2 uses the buffer pool when writing to the logical work file. The number of work files that can be used for sorting is limited only by the buffer pool size when you have the sort assist hardware.

If you do not have the sort hardware, up to 140 logical work files can be allocated per sort, and up to 255 work files can be allocated per user.

It is possible for a sort to complete in the buffer pool without I/Os. This is the ideal situation, but it might be unlikely, especially if the amount of data being sorted is large. The sort row size is actually made up of the columns being sorted (the sort key length) and the columns the user selects (the sort data length). Having a very large virtual pool for sort activity can help avoid disk I/Os—consider using data spaces if your DB2 subsystem runs on a z900 server. (See “Advantages of data spaces” on page 553 for more details.)

When your application needs to sort data, the work files are allocated on a least recently used basis for a particular sort. For example, if five logical work files (LWFs) are to be used in the sort, and the installation has three work file table spaces (WFTSs) allocated, then:

- LWF 1 would be on WFTS 1.
- LWF 2 would be on WFTS 2.
- LWF 3 would be on WFTS 3.
- LWF 4 would be on WFTS 1.
- LWF 5 would be on WFTS 2.

To support large sorts, DB2 can allocate a single logical work file to several physical work file table spaces.

## Improving the performance of sort processing

The following factors affect the performance of DB2 sort processing:

- The larger the sort pool, the more efficient the sort is.
- Minimize I/O contention on the I/O paths to the physical work files. Also, make sure that physical work files are allocated on different I/O paths and packs to minimize I/O contention.
- Allocate additional physical work files in excess of the defaults, and put those work files in their own buffer pool.

  Segregating work file activity enables you to better monitor and tune sort performance. It also allows DB2 to handle sorts more efficiently because these buffers are available only for sort without interference from other DB2 work.

  Applications using created temporary tables use work file space until a COMMIT or ROLLBACK occurs. (If a cursor is defined WITH HOLD, then the data is held past the COMMIT.) If sorts are happening concurrently with the temporary table's existence, then you probably need more space to handle the additional use of the work files.

  For information about defining additional work file table spaces, refer to "Create additional work file table spaces" on page 541.
- Applications should only sort those columns that need to be sorted, as these key fields appear twice in the sort row size. The smaller the sort row size, the more rows can fit it.
- Because varying-length columns are padded to their maximum length, do not select VARCHAR columns unless they are required.

Other factors that influence sort performance include the following:

- The better sorted the data is, the more efficient the sort is.
- If the buffer pool deferred write threshold (DWQT) or data set deferred write threshold (VDWQT) are reached, writes are scheduled. For a large sort using many logical work files, this is difficult to avoid, even if a very large buffer pool is specified.
- If I/Os occur in the sorting process, in the merge phase DB2 uses sequential prefetch to bring pages into the buffer pool with a prefetch quantity of eight pages. However, if the buffer pool is constrained, then DB2 uses a prefetch quantity of four pages or less, or disables prefetch entirely because of the unavailability of enough pages.
- If your DB2 subsystem is running on a processor that has the sort facility hardware instructions, you will see an improvement in the performance of SQL

statements that contain any of the following: ORDER BY clause, GROUP BY clause, CREATE INDEX statement, DISTINCT clause of fullselect, and joins and queries that use sort.

For any SQL statement that initiates sort activity, the DB2 PM SQL activity reports provide information on the efficiency of the sort involved.

# Chapter 28. Improving resource utilization

When system resources are shared among transactions, end user queries, and batch programs, it is important to control how those resources are used. You need to separate data and set priorities carefully. You might choose to emphasize resource use, performance, concurrency, or data security.

Choose the controls that best match your goals. You may, for example, want to minimize resource usage, maximize throughput or response time, ensure a certain level of service to some users, or avoid conflicts between users. Your goal might be to favor a certain class of users or to achieve the best overall system performance.

The number of I/Os and the I/O elapsed times are also important performance considerations in a database system. When you design or tune your database, you should optimize the number of I/Os by using an efficient buffer pool design, and minimize I/O elapsed times by carefully selecting the placement of the DB2 data sets.

Many of the things you currently do for a single DB2 to improve response time or reduce processor consumption also hold true in the data sharing environment. Thus, most of the information in this chapter holds true for data sharing as well. For more information about tuning in a data sharing environment, see Chapter 6 of *DB2 Data Sharing: Planning and Administration*.

This chapter covers the following topics:

- "Controlling resource usage"
- "Resource limit facility (governor)" on page 581
- "Managing the opening and closing of data sets" on page 593
- "Planning the placement of DB2 data sets" on page 597
- "DB2 logging" on page 599
- "Improving disk utilization: space and device utilization" on page 606
- "Improving main storage utilization" on page 609
- "Performance and the storage hierarchy" on page 611
- "MVS performance options for DB2" on page 614

## Controlling resource usage

DB2 includes a resource limit facility (governor), which helps control the use of DB2 resources. Other facilities, such as MVS workload management (SRM and WLM), and the QMF governor, complement the DB2 governor. Because DB2 is integrated with the operating system and transaction subsystems, control definitions are used in most cases. This integration simplifies the task of controlling resources for the user.

Each of the objectives presented in Table 76 on page 580 is matched with a control facility that you can use to achieve the objective.

*Table 76. Controlling the use of resources*

| Objective | How to accomplish it | Where it is described |
|---|---|---|
| Prioritize resources | MVS workload management | “Prioritize resources”, “MVS performance options for DB2” on page 614 and “Using Workload Manager to set performance objectives” on page 629 |
| Limit resources for each job | Time limit on job or step (through MVS or JCL) | “Limit resources for each job” |
| Limit resources for TSO sessions | Time limit for TSO logon | “Limit resources for TSO sessions” on page 581 |
| Limit resources for IMS and CICS | IMS and CICS controls | “Limit resources for IMS and CICS” on page 581 |
| Limit resources for a stored procedure | ASUTIME column of SYSIBM.SYSROUTINES catalog table. | “Limit resources for a stored procedure” on page 581 |
| Limit dynamic statement execution time | QMF governor and DB2 resource limit facility | “Resource limit facility (governor)” on page 581 |
| Reduce locking contention | DB2 locking parameters, DISPLAY DB LOCKS, lock trace data, database design | “Chapter 30. Improving concurrency” on page 643 |
| Evaluate long-term resource usage | Accounting trace data, DB2 PM reports | “DB2 Performance Monitor (DB2 PM)” on page 1039 |
| Predict resource consumption | DB2 EXPLAIN statement, Visual Explain, DB2 Estimator, predictive governing capability | “Chapter 33. Using EXPLAIN to improve SQL performance” on page 789 and “Predictive governing” on page 589 |
| Control use of parallelism | DB2 resource limit facility, SET CURRENT DEGREE statement | “Disabling query parallelism” on page 854 |

## Prioritize resources

The **OS/390 WorkLoad Manager (WLM)** controls the execution of DB2 work based on the priorities that you set. See *OS/390 MVS Initialization and Tuning Guide* for more information about setting priorities on work.

In CICS environments, DB2 work is performed in subtasks; therefore, the work is managed at that level. You can set the priority of the DB2 work relative to the CICS main task through the resource control table.

In other environments such as batch and TSO, which typically have a single task requesting DB2 services, the task-level processor dispatching priority is irrelevant. Access to processor and I/O resources for synchronous portions of the request is governed solely by OS/390 workload manager.

## Limit resources for each job

Because most of the resource usage occurs within the standard job structure, you can control processor usage by changing the TIME parameter for the job or step. The time limit applies even when DB2 is sorting the result rows. If the time limit is exceeded, the job step abends, and any uncommitted work is rolled back. If you want to control the total amount of resources used, rather than the amount used by a single query, then use this control.

Refer to the *OS/390 MVS JCL User's Guide* for more information on setting resource limits.

## Limit resources for TSO sessions

Time limits can apply to either TSO sessions or to batch jobs. Your MVS system programmer can provide a time parameter on the logon procedure or on a job statement in the logon preprompt exit. This time limit is for the session, rather than for an individual query or a single program. If you want to control the amount of resources used for an entire TSO session, rather than the amount used by a single query, then use this control.

You can find more information about setting the resource limit for a TSO session in these manuals:

- *OS/390 TSO/E Programming Guide*
- *OS/390 TSO/E Customization*

## Limit resources for IMS and CICS

You can use various IMS commands (including PROCLIM, or processing limit) to limit the resources used by a transaction, a class, a program, a database, or a subsystem. For more information, see *IMS Command Reference*.

For a detailed description of performance factors in a CICS system, see *CICS/ESA Performance Guide*.

## Limit resources for a stored procedure

DB2 stored procedures are especially designed for high volume online transactions. To establish limits for stored procedures, you can:

- Set a processor limit for each stored procedure, by updating the ASUTIME column of the SYSIBM.SYSROUTINES catalog table. This limit allows DB2 to cancel procedures that loop.
- Set a limit for the number of times a procedure can terminate abnormally, by specifying a value in the MAX ABEND COUNT field on installation panel DSNTIPX. This limit prevents a problem procedure from overwhelming the system with abend dump processing.

For information about controlling the amount of storage used by stored procedures address spaces, see “Controlling address space storage” on page 874.

# Resource limit facility (governor)

DB2’s resource limit facility (governor) lets you perform the following activities:

- Set warning and error thresholds by which the governor can inform users (via your application programs) that a certain processing limit might be exceeded for a particular dynamic SELECT, INSERT, UPDATE, or DELETE statement. This is called *predictive governing*.
- Stop a currently executing dynamic SQL statement (SELECT, INSERT, UPDATE, or DELETE) that exceeds the processor limit that you have specified for that statement. This is sometimes called *reactive governing*, to differentiate its function from that of *predictive* governing, a function that is also handled by the

resource limit facility. The resource limit facility does not control static SQL statements whether or not they are executed locally or remotely.

- Restrict bind and rebind activities to avoid performance impacts on production data.
- Restrict particular parallelism modes for *dynamic* queries.

***Data sharing:*** See Chapter 6 of *DB2 Data Sharing: Planning and Administration* for information about special considerations for using the resource limit facility in a data sharing group.

This section includes the following topics:

- "Using resource limit tables (RLSTs)"
- "Governing dynamic queries" on page 587
- "Restricting bind operations" on page 592
- "Restricting parallelism modes" on page 592

## Using resource limit tables (RLSTs)

Use one or more *resource limit specification tables* (RLSTs) to give governing information to DB2.

If you are a system administrator, you must determine how your location intends to use the governor and create several local procedures. Establish a procedure for creating and maintaining your RLSTs, and for establishing limits for any newly written applications. Develop procedures for console operators, such as switching RLSTs every day at a certain time.

### Creating an RLST

Resource limit specification tables can reside in any database; however, because a database has some special attributes while the resource limit facility is active, it is best to put RLSTs in their own database.

When you install DB2, installation job DSNTIJSG creates a database, table space, table, and descending index for the resource limit specification. You can tailor those statements. For more information about job DSNTIJSG, see Part 2 of *DB2 Installation Guide*.

To create a new resource limit specification table, use the following statements, also included in installation job DSNTIJSG. You must have sufficient authority to define objects in the DSNRLST database and to specify *authid*, which is the authorization ID specified on field RESOURCE AUTHID of installation panel DSNTIPP.

***Creating the table:*** Use the following statement:

```
CREATE TABLE authid.DSNRLSTxx
       (AUTHID   CHAR(8)      NOT NULL WITH DEFAULT,
        PLANNAME CHAR(8)      NOT NULL WITH DEFAULT,
        ASUTIME  INTEGER,
        -------3-column format --------
        LUNAME   CHAR(8)      NOT NULL WITH DEFAULT,
        -------4-column format --------
        RLFFUNC  CHAR(1)      NOT NULL WITH DEFAULT,
        RLFBIND  CHAR(1)      NOT NULL WITH DEFAULT,
        RLFCOLLN CHAR(18)     NOT NULL WITH DEFAULT,
        RLFPKG   CHAR(8)      NOT NULL WITH DEFAULT),
        -------8-column format --------
        RLFASUERR  INTEGER,
        RLFASUWARN INTEGER,
        RLF_CATEGORY_B CHAR(1) NOT NULL WITH DEFAULT)
        -------11-column format --------
         IN DSNRLST.DSNRLSxx;
```

The name of the table is *authid*.DSNRLST*xx*, where *xx* is any 2-character alphanumeric value, and *authid* is specified when DB2 is installed. Because the two characters *xx* must be entered as part of the START command, they must be alphanumeric—no special or DBCS characters.

All future column names defined by IBM will appear as RLF*xxxxx*. To avoid future naming conflicts, begin your own column names with characters other than RLF.

***Creating the index:*** To create an index for the 11-column format, use the following SQL:

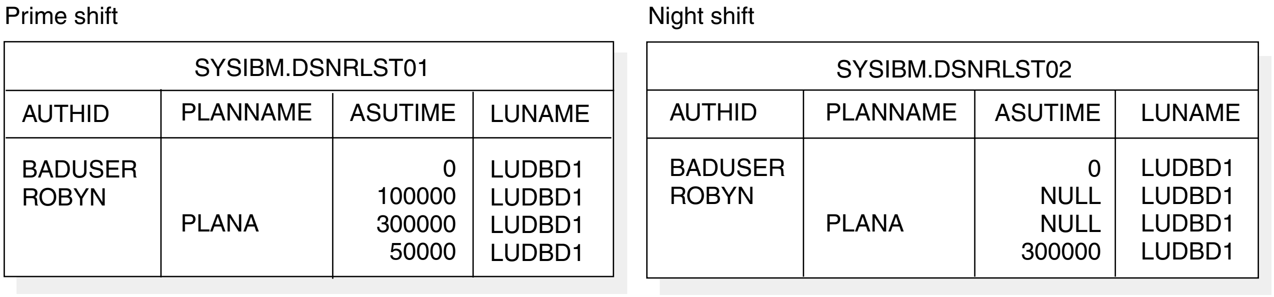
```
CREATE UNIQUE INDEX authid.DSNARLxx
       ON authid.DSNRLSTxx
         (RLFFUNC, AUTHID DESC, PLANNAME DESC,
          RLFCOLLN DESC, RLFPKG DESC, LUNAME DESC)
         CLUSTER CLOSE NO;
```

The *xx* in the index name (DSNARL*xx*) must match the *xx* in the table name (DSNRLST*xx*) and it must be a descending index.

***Populating the RLST:*** Use the SQL statements INSERT, UPDATE, and DELETE to populate the resource limit specification table. The limit that exists when a job makes its first dynamic SELECT, INSERT, UPDATE, or DELETE statement applies throughout the life of the job. If you update the resource limit specification table while a job is executing, that job’s limit does not change; instead, the updates are effective for all new jobs and for those that have not issued their first dynamic SELECT, INSERT, UPDATE, or DELETE statement.

To insert, update, or delete from the resource limit specification table, you need only the usual table privileges on the RLST. No higher authority is required.

***Starting and stopping the RLST:*** Activate any particular RLST by using the DB2 command START RLIMIT ID=*xx* where *xx* is the 2-character identifier you specified on the name DSNRLSTxx. This command gives you the flexibility to use a different RLST for prime shift than you do for evening shift, as in Figure 66; however, only one can be active at a time. At installation time, you can specify a default RLST to be used each time DB2 is restarted. For more information on resource limit facility subsystem parameters, see Part 2 of *DB2 Installation Guide*.



Prime shift

| SYSIBM.DSNRLST01 | | | |
|---|---|---|---|
| AUTHID | PLANNAME | ASUTIME | LUNAME |
| BADUSER | | 0 | LUDBD1 |
| ROBYN | | 100000 | LUDBD1 |
| | PLANA | 300000 | LUDBD1 |
| | | 50000 | LUDBD1 |

Night shift

| SYSIBM.DSNRLST02 | | | |
|---|---|---|---|
| AUTHID | PLANNAME | ASUTIME | LUNAME |
| BADUSER | | 0 | LUDBD1 |
| ROBYN | | NULL | LUDBD1 |
| | PLANA | NULL | LUDBD1 |
| | | 300000 | LUDBD1 |

*Figure 66. Examples of RLST for day and night shifts. During the night shift, AUTHID ROBYN and all PLANA users from LUDBD1 run without limit.*

If the governor is active and you restart it without stopping it, any jobs that are active continue to use their original limits, and all new jobs use the limits in the new table.

If you stop the governor while a job is executing, the job runs with *no limit*, but its processing time continues to accumulate. If you later restart the governor, the new limit takes effect for an active job only when the job passes one of several internal checkpoints. A typical dynamic statement, which builds a result table and fetches from it, passes those checkpoints at intervals that can range *from moments to hours*. As a result, your change to the governor might not stop an active job within the time you expect.

Use the DB2 command CANCEL THREAD to stop an active job that does not pick up the new limit when you restart the governor.

***Restricted activity on the RLST:*** While the governor is active, you cannot execute the following SQL statements on the RLST, or the table space and database in which the RLST is contained:

- DROP DATABASE
- DROP INDEX
- DROP TABLE
- DROP TABLESPACE
- RENAME TABLE

You cannot stop a database or table space that contains an active RLST; nor can you start the database or table space with ACCESS(UT).

## Descriptions of the RLST columns

Here is a complete description for all RLST columns. In no case will all columns in a particular row be populated; the columns you must populate depend on what function is performed by that row (determined by the value in RLFFUNC) and how narrowly you want to qualify values. For example, you can qualify broadly by leaving the AUTHID column blank, which means that the row applies to all authorization IDs. Or, you can qualify very narrowly by specifying a different row for each authorization ID for which the function applies.

***Search order:*** DB2 tries to find the most exact match when it determines which row to use for a particular function. The search order depends on which function is being requested (type of governing, bind operations, or parallelism restrictions). The search order is described under each of those functions.

**AUTHID**

The resource specification limits apply to this primary authorization ID. A blank means that the limit specifications in this row apply to all authorization IDs for the location that is specified in LUNAME.

**PLANNAME**

The resource specification limits apply to this plan. If you are specifying a function that applies to plans (RLFFUNC=blank or '6'), a blank means that the limit specifications in this row apply to all plans for the location that is specified in LUNAME. Qualify by plan name only if the dynamic statement is issued from a DBRM bound in a plan, not a package; otherwise, DB2 does not find this row. If the RLFFUNC column contains a function for packages ('1,' '2,' or '7'), this column must be blank; if it is not blank, the row is ignored.

**ASUTIME**

The number of processor service units allowed for any single dynamic SELECT, INSERT, UPDATE, or DELETE statement. Use this column for reactive governing.

Other possible values and their meanings are:

**null** No limit

**0 (zero) or a negative value**

No dynamic SELECT, INSERT, UPDATE, or DELETE statements are permitted.

The governor samples the processing time in service units. Service units are independent of processor changes. The processing time for a particular SQL statement varies according to the processor on which it is executed, but the service units required remains roughly constant. The service units consumed are not exact between different processors because the calculations for service units are dependent on measurement averages performed before new processors are announced. A relative metric is used so that the RLST values do not need to be modified when processors are changed. However, in some cases, DB2 workloads can differ from the measurement averages. In these cases, RLST value changes may be necessary. For information about how to calculate service units, see “Calculating service units” on page 591.

**LUNAME**

The LU name of the location where the request originated. A blank value in this column represents the local location, *not* all locations. The value PUBLIC represents all of the DBMS locations in the network; these locations do not need to be DB2 subsystems. PUBLIC is the only value for TCP/IP connections.

**RLFFUNC**

Specifies how the row is used. The values that have an effect are:

**blank** The row reactively governs dynamic SELECT, INSERT, UPDATE, or DELETE statements by plan name.

**’1’** The row reactively governs bind operations.

**’2’** The row reactively governs dynamic SELECT, INSERT, UPDATE, or DELETE statements by package or collection name.

**’3’** The row disables query I/O parallelism.

**’4’** The row disables query CP parallelism.

**’5’** The row disables Sysplex query parallelism.

**’6’** The row predictively governs dynamic SELECT, INSERT, UPDATE, or DELETE statements by plan name.

**’7’** The row predictively governs dynamic SELECT, INSERT, UPDATE, or DELETE statements by package or collection name.

All other values are ignored.

**RLFBIND**

Shows whether bind operations are allowed. An 'N' implies that bind operations are not allowed. Any other value means that bind operations are allowed. This column is used only if RLFFUNC is set to '1'.

**RLFCOLLN**

Specifies a package collection. A blank value in this column means that the row applies to all package collections from the location that is specified in LUNAME. Qualify by collection name only if the dynamic statement is issued from a package; otherwise DB2 does not find this row. If RLFFUNC=blank, '1,' or '6', then RLFCOLLN must be blank.

**RLFPKG**

Specifies a package name. A blank value in this column means that the row applies to all packages from the location that is specified in LUNAME. Qualify by package name only if the dynamic statement is issued from a package; otherwise DB2 does not find this row. If RLFFUNC=blank, '1', or '6', then RLFPKG must be blank.

**RLFASUERR**

Used for predictive governing (RLFFUNC= '6' or '7'), and only for statements that are in cost category A. The error threshold number of system resource manager processor service units allowed for a single dynamic SELECT, INSERT, UPDATE, or DELETE statement. If the predicted processor cost (in service units) is greater than the error threshold, an SQLCODE -495 is returned to the application.

Other possible values and their effects are:

**null** No error threshold

**0 (zero) or a negative value**

All dynamic SELECT, INSERT, UPDATE, or DELETE statements receive SQLCODE -495.

**RLFASUWARN**

Used for predictive governing (RELFFUNC= '6' or '7'), and only for statements that are in cost category A. The warning threshold number of processor service units that are allowed for a single dynamic SELECT, INSERT, UPDATE, or DELETE statement. If the predicted processor cost (in service units) is greater than the warning threshold, an SQLCODE +495 is returned to the application.

Other possible values and their effects are:

**null** No warning threshold

**0 (zero) or a negative value**

All dynamic SELECT, INSERT, UPDATE, or DELETE statements receive SQLCODE +495.

**Important:** Make sure the value for RLFASUWARN is less than that for RLFASUERR. If the warning value is higher, the warning is never reported. The error takes precedence over the warning.

**RLF_CATEGORY_B**

Used for predictive governing (RLFFUNC='6' or '7'). Tells the governor the default action to take when the cost estimate for a given statement falls into cost category B, which means that the predicted cost is indeterminate and probably too low. You can tell if a statement is in cost category B by running EXPLAIN and checking the COST_CATEGORY column of the DSN_STATEMNT_TABLE.

The acceptable values are:

**blank** By default, prepare and execute the SQL statement.

**Y** Prepare and execute the SQL statement.

**N** Do not prepare or execute the SQL statement. Return SQLCODE -495 to the application.

**W** Complete the prepare, return SQLCODE +495, and allow the application logic to decide whether to execute the SQL statement or not.

## Governing dynamic queries

You can use two modes of governing to control the amount of system resources that a dynamic SELECT, INSERT, UPDATE, or DELETE uses. These modes are called *reactive* and *predictive*. You can use either mode or both together.

***Specifying reactive governing:*** Specify either of the following values in the RLFFUNC column of the RLST:

**blank** Govern by plan name

**’2’** Govern by package name

Any statement that exceeds a limit you set in the RLST terminates with a -905 SQLCODE and a corresponding '57014' SQLSTATE. You can establish a single limit for all users, different limits for individual users, or both. Limits do not apply to primary or secondary authorization IDs with installation SYSADM or installation SYSOPR authority. For queries entering DB2 from a remote site, the local site limits are used.

***Specifying predictive governing:*** Specify either of the following values in the RLFFUNC column of the RLST:

**’6’** Govern by plan name

**’7’** Govern by package name

See “Qualifying rows in the RLST” for more information about how to qualify rows in the RLST. See “Predictive governing” on page 589 for more information about using predictive governing.

This section includes the following topics:

- “Qualifying rows in the RLST”
- “Predictive governing” on page 589
- “Combining reactive and predictive governing” on page 590
- “Governing statements from a remote site” on page 591
- “Calculating service units” on page 591

### Qualifying rows in the RLST

DB2 tries to find the closest match when determining limits for a particular dynamic statement. In summary, the search order of matching is as follows:

1. Exact match
2. Authorization ID
3. Plan name, or collection name and package name
4. LU name
5. No row match

***Governing by plan or package name:*** Governing by plan name and package name are mutually exclusive.

- Plan name

  The RLF governs the DBRMs in the MEMBER list specified on the BIND PLAN command. The RLFFUNC, RLFCOLLN, and RLFPKG columns must contain blanks. For example:

*Table 77. Qualifying rows by plan name*

| RLFFUNC | AUTHID | PLANNAME | LUNAME | ASUTIME |
|---|---|---|---|---|
| (blank) | JOE | PLANA | (blank) | (null) |
| (blank) | (blank) | WSPLAN | SAN_JOSE | 15000 |
| (blank) | (blank) | (blank) | PUBLIC | 10000 |

  The first row in Table 77 shows that when Joe runs PLANA at the local location, there are no limits for any dynamic statements in that plan.

  The second row shows that if anyone runs WSPLAN from SAN_JOSE, the dynamic statements in that plan are restricted to 15000 SUs each.

  The third row is entered as a cap for any unknown authorization IDs or plan names from any location in the network, including the local location. (An alternative would be to let the default values on installation panel DSNTIPR and DSNTIPO serve as caps.)

- Collection and package name

  The RLF governs the packages used during the execution of the SQL application program. PLANNAME must contain blank, and RLFFUNC must contain ’2’.

*Table 78. Qualifying rows by collection or package name*

| RLFFUNC | AUTHID | RLFCOLLN | RLFPKG | LUNAME | ASUTIME |
|---|---|---|---|---|---|
| 2 | JOE | COLL1 | (blank) | (blank) | 40000 |
| 2 | (blank) | (blank) | DSNESPCS | PUBLIC | 15000 |

  The first row in Table 78 shows that when Joe runs any package in collection 1 from the local location, dynamic statements are restricted to 40000 SUs.

  The second row indicates that if anyone from any location (including the local location) runs SPUFI package DSNESPCS, dynamic statements are limited to 15000 SUs.

***Governing by LU name:*** Specify an originating system’s LU name in the LUNAME column, or, specify PUBLIC for all remote LUs. An LUNAME with a value other than PUBLIC takes precedence over PUBLIC. If you leave LUNAME blank, DB2 assumes that you mean the local location only and none of your incoming distributed requests will qualify. PUBLIC is the only value for TCP/IP connections.

***Setting a default for when no row matches:*** If no row in the RLST matches the currently executing statement, then DB2 uses the default set on the RLST ACCESS ERROR field of installation panel DSNTIPO (for queries that originate locally) or DSNTIPR (for queries that originate remotely). This default applies to reactive governing only.

For predictive governing, if no row matches, then there is no predictive governing.

## Predictive governing

DB2's predictive governing capability has an advantage over the reactive governor in that it avoids wasting processing resources by giving you the ability to prevent a query from running when it appears that it exceeds processing limits. With the reactive governor, those resources are already used before the query is stopped.

See Figure 67 for an overview of how predictive governing works.

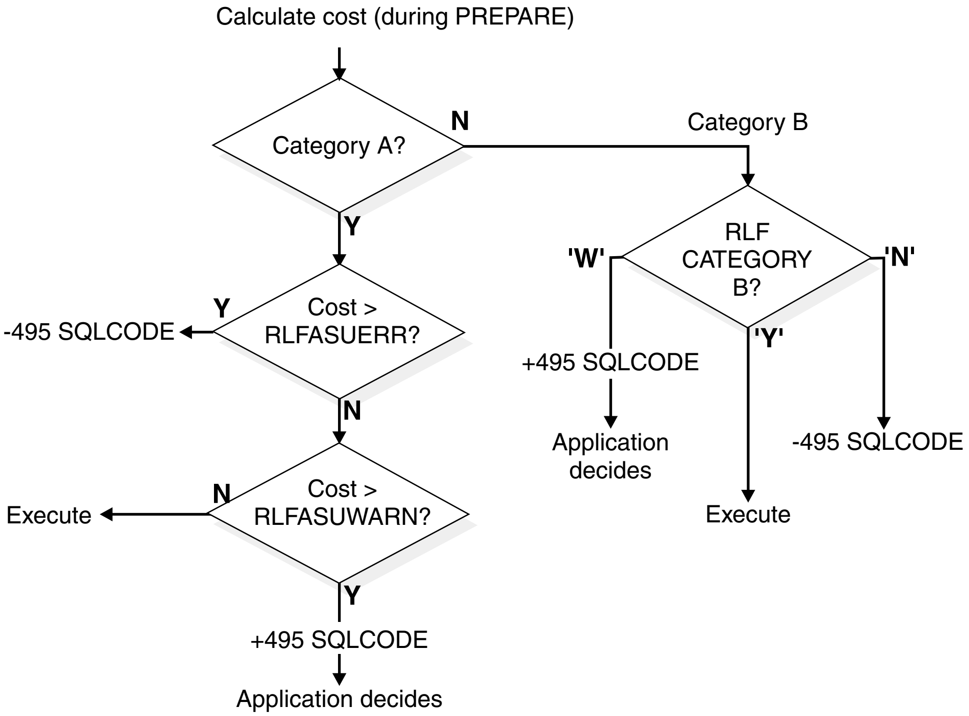


*Figure 67. Processing for predictive governing*

At prepare time for a dynamic SELECT, INSERT UPDATE, or DELETE statement, DB2 searches the active RLST to determine if the processor cost estimate exceeds the error or warning threshold that you set in RLFASUWARN and RLFASUERR columns for that statement. DB2 compares the cost estimate for a statement to the thresholds you set, and the following actions occur:

- If the cost estimate is in cost category A and the error threshold is exceeded, DB2 returns a -495 SQLCODE to the application, and the statement is not prepared or run.
- If the estimate is in cost category A and the warning threshold is exceeded, a +495 SQLCODE is returned at prepare time. The prepare is completed, and the application or user decides whether to run the statement.
- If the estimate is in cost category B, DB2 takes the action you specify in the RLF_CATEGORY_B column; that is, it either prepares and executes the statement, does not prepare or execute the statement, or returns a warning SQLCODE, which lets the application decide what to do.
- If the estimate is in cost category B and the warning threshold is exceeded, a +495 SQLCODE is returned at prepare time. The prepare is completed, and the application or user decides whether to run the statement.

***Example:*** Table 79 on page 590 is an RLST with two rows that use predictive governing.

Table 79. Predictive governing example

| RLFFUNC | AUTHID | RLFCOLLN | RLFPKG | RLFASUWARN | RLFASUERR | RLF_ CATEGORY_ B |
|---|---|---|---|---|---|---|
| 7 | (blank) | COLL1 | C1PKG1 | 900 | 1500 | Y |
| 7 | (blank) | COLL2 | C2PKG1 | 900 | 1500 | W |

The rows in the RLST for this example cause DB2 to act as follows for all dynamic INSERT, UPDATE, DELETE, and SELECT statements in the packages listed in this table (C1PKG1 and C2PKG1):

- Statements in cost category A that are predicted to be less than 900 SUs will execute.
- Statements in cost category A that are predicted to be between 900 and 1500 SUs receive a +495 SQLCODE.
- Statements in cost category A that are predicted to be greater than 1500 SUs receive SQLCODE -495, and the statement is not executed.

***Cost category B:*** The two rows differ only in how statements in cost category B are treated. For C1PKG1, the statement will execute. For C2PKG2, the statements receive a +495 SQLCODE and the user or application must decide whether to execute the statement.

## Combining reactive and predictive governing

A dynamic statement can be governed both before and after the statement is executed. For example, if the processing cost estimate is in cost category B and you decide to run the statement, you can use the RLST to terminate the statement after a certain amount of processor time, the same as it does today. To use both modes of governing, you need two rows in the RLST as shown in Table 80.

Table 80. Combining reactive and predictive governing

| RLFFUNC | AUTHID | PLANNAME | ASUTIME | RLFASUWARN | RLFASUERR | RLF_ CATEGORY_ B |
|---|---|---|---|---|---|---|
| 6 | USER1 | PLANA | 0 | 800 | 1000 | W |
| (blank) | USER1 | PLANA | 1100 | 0 | 0 | (blank) |

The rows in the RLST for this example cause DB2 to act as follows for a dynamic SQL statement that runs under PLANA:

***Predictive mode:***

- If the statement is in COST_CATEGORY A and the cost estimate is greater than 1000 SUs, USER1 receives SQLCODE -495 and the statement is not executed.
- If the statement is in COST_CATEGORY A and the cost estimate is greater than 800 SUs but less than 1000 SUs, USER1 receives SQLCODE +495.
- If the statement is in COST_CATEGORY B, USER1 receives SQLCODE +495.

***Reactive mode:*** In either of the following cases, a statement is limited to 1100 SUs:

- The cost estimate for a statement in COST_CATEGORY A is less than 800 SUs

- The cost estimate for a COST_CATEGORY A is greater than 800 and less than 1000 or is in COST_CATEGORY B and the user chooses to execute the statement

## Governing statements from a remote site

For distributed processing, keep in mind the following guidelines:

- For dynamic statements coming from requesters using DRDA protocols, you must govern by package name (RLFFUNC=2 or RLFFUNC=7), which means that PLANNAME must be blank. Specify the originating system's LU name in the LUNAME column, or specify PUBLIC for all remote LUs. If you leave LUNAME blank, DB2 assumes that you mean the local location only and none of your incoming distributed requests will qualify.
- For dynamic statements coming from requesters using DB2 private protocol, you must govern by plan name (RLFFUNC=(blank) or '6'), which means that RLFCOLLN and RLFPKG must be blank. Specify the originating system's LU name in the LU column, or specify PUBLIC for all remote LUs. Again, a value other than PUBLIC takes precedence over PUBLIC. PLANNAME can be blank or the name of the plan created at the requester's location. RLFPKG and RLFCOLLN must be blank.
- For dynamic statements coming from requesters using TCP/IP, you cannot specify the LU name. You must use PUBLIC.
- If no row is present in the RLST to govern access from a remote location, the limit is the default set on the RLST ACCESS ERROR field of installation panel DSNTIPR.

## Calculating service units

The easiest way to get SU times for the ASUTIME, RLFASUWARN, or RLFASUERR columns is to use the value in the PROCSU column of DSN_STATEMNT_TABLE as your starting point. You can also get the value from the IFCID 0022 record.

However, if you don't have statement table information or if there are adhoc queries for which you have no information, you can use the following formula to calculate SU time:

```
SU time = processor time × service units per second value
```

The value for service units per second depends on the processor model. You can find this value for your processor model in *OS/390 MVS Initialization and Tuning Guide*, where SRM is discussed.

For example, if processor A is rated at 900 service units per second and you do not want any single dynamic SQL statement to use more than 10 seconds of processor time, you could set ASUTIME as follows:

```
ASUTIME time = 10 seconds × 900 service units/second = 9000 service units
```

Later, you could upgrade to processor B, which is rated at 1000 service units per second. If the value you set for ASUTIME remains the same (9000 service units), your dynamic SQL is only allowed 9 seconds for processing time but an equivalent number of processor service units:

```
ASUTIME = 9 seconds × 1000 service units/second = 9000 service units
```

As this example illustrates, after you establish an ASUTIME (or RLFASUWARN or RLFASUERR) for your current processor, there is no need to modify it when you change processors.

## Restricting bind operations

To restrict bind operations, use RLF function 1 (RLFFUNC=’1’) and qualify rows by authorization ID and LU name. The same precedence search order applies:

1. AUTHID and LUNAME match
2. AUTHID matches
3. LUNAME matches

   A value of PUBLIC for LUNAME applies to all authorization IDs at all locations, while a blank LUNAME governs bind operations for IDs at the local location only.
4. If no entry matches, or if your RLST cannot be read, then the resource limit facility does not disable bind operations.

### Example

Table 81 is an example of an RLST that disables bind operations for all but three authorization IDs. Notice that BINDER from the local site is able to bind but that BINDER from San Francisco is not able to bind. Everyone else from all locations, including the local one, is disabled from doing binds.

*Table 81. Restricting bind operations*

| RLFFUNC | AUTHID | LUNAME | RLFBIND |
|---|---|---|---|
| 1 | BINDGUY | PUBLIC | |
| 1 | NIGHTBND | PUBLIC | |
| 1 | (blank) | PUBLIC | N |
| 1 | BINDER | SANFRAN | N |
| 1 | BINDER | (blank) | |

## Restricting parallelism modes

The RLST lets you disable parallelism for dynamic queries. You might want to do this if, for example, one mode of parallelism for a particular query performs better or one mode for a particular query does not perform well. For governing query parallelism, remember the following guidelines:

- The resource limit facility only governs query parallelism for dynamic queries when the value of the CURRENT DEGREE special register is 'ANY'.
- To disable all query parallelism for a dynamic query, you need rows in your RLST to cover all possible modes of parallelism for your system. You need one row with RLFFUNC='3' and one row with RLFFUNC='4', and, if Sysplex query parallelism is possible in your system, then you must add an additional row containing RLFFUNC='5'. If parallelism is disabled for a query, the query runs sequentially.
- Qualifying by plan or by package are not separate functions as they are for predictive and reactive governing. If you want to qualify by plan name, the query must be executed from a plan or DB2 will not find the row. If you want to qualify by package name, the query must be executed from a package or DB2 will not find the row.

  If you want to qualify rows by authorization ID only, you can leave all three columns blank: RLFCOLLN, RLFPKG, and RLFPLAN.
- If no entry can be found in your RLST that applies to parallelism, or if your RLST cannot be read, then the resource limit facility does not disable query parallelism.

*Table 82. Example RLST to govern query parallelism*

| RLFFUNC | AUTHID | LUNAME | RLFCOLLN | RLFPKG |
|---|---|---|---|---|
| 3 | (blank) | PUBLIC | blank | IOHOG |
| 4 | (blank) | PUBLIC | blank | CPUHOG |
| 5 | (blank) | PUBLIC | blank | CPUHOG |

If the RLST in Table 82 is active, it causes the following effects:

- Disables I/O parallelism for all dynamic queries in IOHOG.
- Disables CP parallelism and Sysplex query parallelism for all dynamic queries in CPUHOG.

## Managing the opening and closing of data sets

| Having the needed data sets open and available for use is important for the | performance of transactions. However, the number of open data sets affects the | amount of available storage, and number of open data sets in read-write state | affects restart time. This section describes how DB2 manages this open and close activity and gives some recommendations about how you can influence this processing.

This section includes the following topics:

- "Determining the maximum number of open data sets"
- "Understanding the CLOSE YES and CLOSE NO options" on page 595
- "Switching to read-only for infrequently updated page sets" on page 596

## Determining the maximum number of open data sets

DB2 defers closing and deallocating the table spaces or indexes until the number of open data sets reaches one of the following limits:

- The MVS limit for the number of concurrently open data sets.
- 99% of the value that you specified for DSMAX.

  When DSMAX is reached, DB2 closes a number of data sets not in use equal to 3% of the value DSMAX. Thus, DSMAX controls not only the limit of open data sets, but also the number of data sets that are closed when that limit is reached.

### How DB2 determines DSMAX

Initially, DB2 calculates DSMAX as follows:

- Let *concdb* be the number of concurrent databases specified on installation panel DSNTIPE.
- Let *tables* be the number of tables per database specified on installation panel DSNTIPD.
- Let *indexes* be the number of indexes per table. The installation CLIST sets this variable to 2.
- Let *tblspaces* be the number of table spaces per database specified on installation panel DSNTIPD.

DB2 calculates the number of open data sets with the following formula:

*concdb* × {(*tables* × *indexes*) + *tblspaces*}

### Modifying DSMAX

The formula used by DB2 does not take partitioned or LOB table spaces into account. Those table spaces can have many data sets. If you have many partitioned table spaces or LOB table spaces, you might need to increase DSMAX. Don't forget to consider the data sets for nonpartitioning indexes defined on partitioned table spaces. If those indexes are defined with a small PIECESIZE, there could be many data sets. You can modify DSMAX by updating field DSMAX - MAXIMUM OPEN DATA SETS on installation panel DSNTIPC.

***Calculating the size of DSMAX:*** To reduce the open and close activity of data sets, it is important to set DSMAX correctly. DSMAX should be larger than the maximum number of data sets that are open and in use at one time. For the most accurate count of open data sets, refer to the OPEN/CLOSE ACTIVITY section of the DB2 PM statistics report. Make sure the statistics trace was run at a peak period, so that you can obtain the most accurate maximum figure.

To calculate the total number of data sets (rather than the number that are open during peak periods), you can do the following:

1. To find the number of simple and segmented table spaces and the accompanying indexes, add the results of the following two queries. These calculations assume that you have one data set per each simple, segmented, or LOB table space, and one data set per nonpartitioning index. Adjust accordingly if you have more than that.

   These catalog queries are included in DSNTESP in SDSNSAMP. You can use them as input to SPUFI.

   **General-use Programming Interface**

   **Query 1**

   ```
   SELECT CLOSERULE, COUNT(*)
    FROM SYSIBM.SYSTABLESPACE
    WHERE PARTITIONS < 1
    GROUP BY CLOSERULE;
   ```

   **End of General-use Programming Interface**

   **General-use Programming Interface**

   **Query 2**

   ```
   SELECT CLOSERULE, COUNT(*)
    FROM   SYSIBM.SYSINDEXES T1, SYSIBM.SYSINDEXPART T2
    WHERE  T1.NAME = T2.IXNAME
    AND    T1.CREATOR = T2.IXCREATOR
    AND    T2.PARTITION < 1
    GROUP BY CLOSERULE;
   ```

   **End of General-use Programming Interface**

2. To find the number of data sets for partitioned table spaces, use the following query, which returns the number of partitioned table spaces and the number of partitions.

   **General-use Programming Interface**

   **Query 3**

   ```
   SELECT CLOSERULE, COUNT(*), SUM(PARTITIONS)
    FROM SYSIBM.SYSTABLESPACE
    WHERE PARTITIONS > 0
    GROUP BY CLOSERULE;
   ```

**End of General-use Programming Interface**

Partitioned table spaces can require up to 254 data sets for the data, 254 data sets for the partitioning index, and one data set for each piece of the nonpartitioning index.

3. To find the total number of data sets, add:
   - The numbers that result from Query 1 and Query 2
   - Two times the sum of the partitions as obtained from Query 3. (This allows for data partitions *and* indexes.)

These queries give you the number of CLOSE NO and CLOSE YES data sets. While CLOSE NO data sets tend to stay open when they have been opened, they might never be opened. CLOSE YES data sets are open when they are in use, and they stay open for a period of time after they have been used. For more information about how the CLOSE value affects when data sets are closed, see “Understanding the CLOSE YES and CLOSE NO options”.

### Recommendations

As with many recommendations in DB2, you must weigh the cost of performance versus availability when choosing a value for DSMAX. Consider the following:

- For best performance, you should leave enough margin in your specification of DSMAX so that frequently used data sets can remain open after they are no longer referenced. If data sets are opened and closed frequently, such as every few seconds, you can improve performance by increasing DSMAX.
- The number of open data sets on your subsystem that are in read/write state affects checkpoint costs and log volumes. To control how long data sets stay open in a read/write state, specify values for the RO SWITCH CHKPTS and RO SWITCH TIME fields of installation panel DSNTIPN. See “Switching to read-only for infrequently updated page sets” on page 596 for more information.
- Consider segmented table spaces to reduce the number of data sets.

  To reduce open and close activity, you can try reducing the number of data sets by combining tables into segmented table spaces. This approach is most useful for development or end-user systems where there are a lot of smaller tables that can be combined into single table spaces.

## Understanding the CLOSE YES and CLOSE NO options

**General-use Programming Interface**

This section describes how DB2 manages data set closing and how the CLOSE value for a table space or index affects the process of closing an object’s data sets. The term **page set** refers to a table space or index.

### The process of closing

DB2 dynamically manages page sets using two levels of page set closure—logical close and physical close.

***Logical close:*** This occurs when the application has been deallocated from that page set. This is at either commit or deallocation time, depending on the RELEASE(COMMIT/DEALLOCATE) option of the BIND command, and is driven by the use count. When a page set is logically closed, the page set use count is decremented. When the page set use count is zero, the page set is considered not in use; this makes it a candidate for physical close.

***Physical close:*** This happens when DB2 closes and deallocates the data sets for the page set. SYSLGRNX is updated when a table space or an index defined with COPY YES in read/write mode is physically closed.

### When the data sets are closed

As described in “Determining the maximum number of open data sets” on page 593, the number of open data sets determines when it is necessary to close data sets. When DB2 closes data sets, all data sets for a particular table space, index, or partition are closed.

The value you specify for CLOSE determines the *order* in which page sets are closed. When the open data set count becomes greater than 99% of DSMAX, DB2 first closes page sets defined with CLOSE YES. The least recently used page sets are closed first. To do this, DB2 must keep track of page set usage. This least recently used method is effective; you should have substantially fewer CLOSE NO data sets than DSMAX.

If the number of open data sets cannot be limited by closing page sets or partitions defined with CLOSE YES, DB2 then must close page sets or partitions defined with CLOSE NO. The least recently used CLOSE NO data sets are closed first.

Delaying the physical closure of page sets or partitions until necessary is called *deferred close*. Deferred closing of a page set or partition that is no longer being used means that another application or user can access the table space and employ the accompanying indexes without DB2 reopening the data sets. Thus, deferred closing of page sets or partitions can improve your applications’ performance by avoiding I/O processing.

***Recommendation:*** For a table space whose data is continually referenced, in most cases it does not matter whether it is defined with CLOSE YES or CLOSE NO; the data sets remain open. This is also true, but less so, for a table space whose data is not referenced for short periods of time; because DB2 uses deferred close to manage data sets, the data sets are likely to be open when they are used again.

You could find CLOSE NO appropriate for page sets that contain data you do not frequently use but is so performance-critical that you cannot afford the delay of opening the data sets.

If the number of open data sets is a concern, choose CLOSE YES for page sets with many partitions or data sets.

**End of General-use Programming Interface**

## Switching to read-only for infrequently updated page sets

For both CLOSE YES and CLOSE NO page sets, DB2 automatically converts infrequently updated page sets or partitions from read-write to read-only state according to the values you specify in the RO SWITCH CHKPTS and RO SWITCH TIME fields of installation panel DSNTIPN.

RO SWITCH CHKPTS is the number of consecutive DB2 checkpoints since a page set or partition was last updated; the default is 5. RO SWITCH TIME is the amount of elapsed time since a page set or partition was last updated; the default is 10 minutes. If either condition is met, the page set or partition is converted from read-write to read-only state.

***Updating SYSLGRNX:*** For both CLOSE YES and CLOSE NO page sets, SYSLGRNX entries are updated when the page set is converted from read-write state to read-only state. When this conversion occurs for table spaces, the SYSLGRNX entry is closed and any updated pages are externalized to disk. For indexes defined as COPY NO, there is no SYSLGRNX entry, but the updated pages are externalized to disk.

***Performance benefits of read-only switching:*** An infrequently used page set's conversion from read-write to read-only state results in the following performance benefits:

- Improved data recovery performance because SYSLGRNX entries are more precise, closer to the last update transaction commit point. As a result, the RECOVER utility has fewer log records to process.
- Minimized logging activities. Log records for page set open, checkpoint, and close operations are only written for updated page sets or partitions. Log records are not written for read-only page sets or partitions.

***Recommendations for RO SWITCH TIME and RO SWITCH CHKPTS:*** In most cases, the default values are adequate. However, if you find that the amount of R/O switching is causing a performance problem for the updates to SYSLGRNX, consider increasing the value of RO SWITCH TIME, perhaps to 30 minutes.

# Planning the placement of DB2 data sets

To improve performance, plan the placement of DB2 data sets carefully. Concentrate mainly on data sets for system files (especially the active logs), for the DB2 catalog and directory, and for user data and indexes. The objective is to balance I/O activity between different volumes, control units, and channels, which minimizes the I/O elapsed time and I/O queuing.

This section includes the following topics:

- "Estimating concurrent I/O requests"
- "Crucial DB2 data sets"
- "Changing catalog and directory size and location" on page 598
- "Monitoring I/O activity of data sets" on page 598
- "Work file data sets" on page 599

## Estimating concurrent I/O requests

DB2 has a multi-tasking structure in which each user's request runs under a different task control block (TCB). In addition, the DB2 system itself has its own TCBs and SRBs for logging and database writes. When your DB2 system is loaded with data, you can estimate the maximum number of concurrent I/O requests as:

MAX USERS + 600 sequential prefetches + 300 asynchronous writes

This figure is important when you calculate the number of data paths for your DB2 subsystem.

## Crucial DB2 data sets

First, decide which data sets are crucial to DB2's functioning. To gather this information, use the I/O reports from the DB2 performance trace. If these reports are not available, consider these the most important data sets:

For transactions:

- DSNDB01.SCT02 and its index
- DSNDB01.SPT01 and its index
- DSNDB01.DBD01
- DSNDB06.SYSPLAN table space and indexes on SYSPLANAUTH table
- DSNDB06.SYSPKAGE
- Active logs
- Most frequently used user table spaces and indexes

For queries:

- DSNDB01.DBD01
- DSNDB06.SYSPLAN table space and indexes on SYSPLANAUTH
- DSNDB06.SYSPKAGE
- DSNDB06.SYSDBASE table space and its indexes
- DSNDB06.SYSVIEWS table space and the index on SYSVTREE
- Work file table spaces
- QMF system table data sets
- Most frequently used user table spaces and indexes

These lists do not include other data sets that are less crucial to DB2's performance, such as those that contain program libraries, control blocks, and formats. Those types of data sets have their own design recommendations. But check whether the data sets have used secondary allocations. For best performance, stay within the primary allocations.

## Changing catalog and directory size and location

Consider changing the size or location of your DB2 catalog or directory if necessary for your site. See “Appendix F. Using tools to monitor performance” on page 1029 for guidelines on when to do this.

To change the size or location of DB2 catalog or directory data sets for any one of the situations listed above, you must run the RECOVER utility on the appropriate database, or the REORG utility on the appropriate table space. A hierarchy of recovery dependencies determines the order in which you should try to recover data sets. This order is discussed in the description of the RECOVER utility in Part 2 of *DB2 Utility Guide and Reference*.

## Monitoring I/O activity of data sets

The best way to monitor your I/O activity against database data sets is through IFCID 0199 (statistics class 8). This IFCID can give you information such as the average write I/O delay or the maximum delay for a particular data set over the last statistics reporting interval. The same information is also reported in the DISPLAY BUFFERPOOL command with the LSTATS option.

More detailed information is available, with more overhead, with the I/O trace (performance class 4). If you want information about I/O activity to the log and BSDS data sets, use performance class 5.

You can also use RMF to monitor data set activity. SMF record type 42-6 provides activity information and response information for a data set over a time interval.

| These time intervals include the components of I/O time, such as IOS queue time.
| Using RMF incurs about the same overhead as statistics class 8.

For information on how to tune your environment to improve I/O performance, see "Reducing the time needed to perform I/O operations" on page 541 and 530.

## Work file data sets

Work file data sets are used for sorting, for materializing views and nested table expressions, for temporary tables, and for other activities. DB2 does not distinguish or place priorities on these uses of the work file data sets. Excessive activity from one type of use can interfere and detract from the performance of others. It is important to monitor how work files use devices, both in terms of space use and I/O response times.

***More about temporary work file result tables:*** Part 2 of *DB2 Installation Guide* contains information about how to estimate the disk storage required for temporary work file result tables. The storage is similar to that required for regular tables. When a temporary work file result table is populated using an INSERT statement, it uses work file space.

No other process can use the same work file space as that temporary work file table until the table goes away. The space is reclaimed when the application process commits or rolls back, or when it is deallocated, depending which RELEASE option was used when the plan or package was bound.

To best monitor these result tables, keep work files in a separate buffer pool. Use IFCID 0311 in performance trace class 8 to distinguish these tables from other uses of the work file.

# DB2 logging

DB2 logs changes made to data, and other significant events, as they occur. You can find background information on the DB2 log in "Chapter 18. Managing the log and the bootstrap data set" on page 331. When you focus on logging performance issues, remember that the characteristics of your workload have a direct effect on log write performance. Long-running batch jobs that commit infrequently have a lot more data to write at commit than a typical transaction.

Don't forget to consider the cost of reading the log as well. The cost of reading the log directly affects how long a restart or a recovery occurs because DB2 must read the log data before applying the log records back to the table space.

This section includes the following topics:

- "Logging performance issues and recommendations"
- "Log capacity" on page 602
- "Controlling the amount of log data" on page 604

## Logging performance issues and recommendations

This section provides background on logging operations. By understanding the day-to-day activity on the log, you can more effectively pinpoint when problems occur and better understand how to tune for best performance.

### Log writes

Log writes are divided into two categories: synchronous and asynchronous.

***Asynchronous writes:*** Asynchronous writes are the most common. These asynchronous writes occur when data is updated. Before- and after-image records are usually moved to the log output buffer, and control is returned to the application. However, if no log buffer is available, the application must wait for one to become available.

***Synchronous writes:*** Synchronous writes usually occur at commit time when an application has updated data. This write is called 'forcing' the log because the application must wait for DB2 to force the log buffers to disk before control is returned to the application. If the log data set is not busy, all log buffers are written to disk. If the log data set is busy, the requests are queued until it is freed.

***Writing to two logs:*** Dual logging is shown in Figure 68.

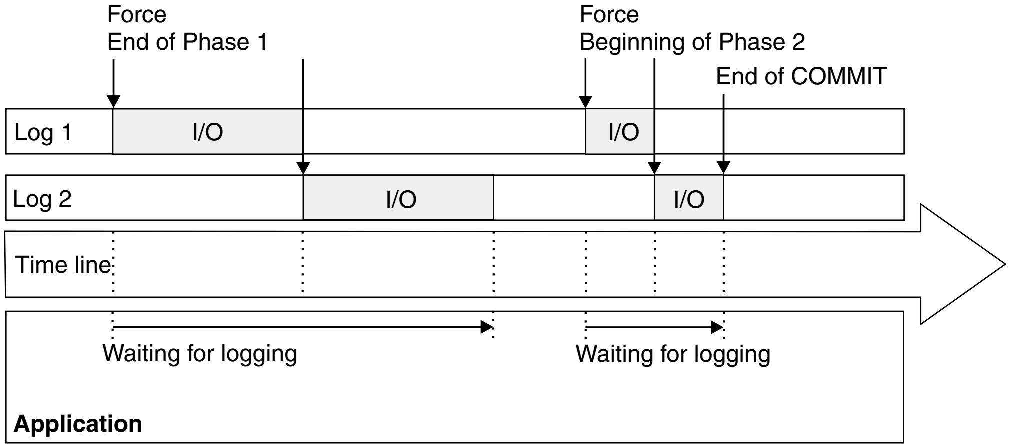


*Figure 68. Dual logging during two-phase commit*

If there are two logs (recommended for availability), the write to the first log, in general, must complete before the write to the second log begins. The first time a log control interval is written to disk, the write I/Os to the log data sets are performed in parallel. However, if the same 4 KB log control interval is again written to disk, then the write I/Os to the log data sets must be done serially to prevent any possibility of losing log data in case of I/O errors on both copies simultaneously.

***Two-phase commit log writes:*** Because they use two-phase commit, applications that use the CICS, IMS, and RRS attachment facilities force writes to the log twice, as shown in Figure 68. The first write forces all the log records of changes to be written (if they have not been written previously because of the write threshold being reached). The second write writes a log record that takes the unit of recovery into an in-commit state.

***Recommendations:*** Recommendations for improving log write performance:

- **Choose a large size for OUTPUT BUFFER size:** The OUTPUT BUFFER field of installation panel DSNTIPL lets you specify the size of the output buffer used for writing active log data sets. The maximum size of this buffer (OUTBUFF) is 400000 KB. Choose as large a size as your system can tolerate to decrease the number of forced I/O operations that occur because there are no more buffers. A large size can also reduce the number of wait conditions. A non-zero value for **D** in Figure 69 on page 601 is an indicator that your output buffer is too small. Ensure that the size you choose is backed up by real storage to avoid paging to expanded storage, which can negatively affect performance.

```
LOG ACTIVITY                QUANTITY  /MINUTE  /THREAD  /COMMIT
--------------------------- --------  -------  -------  -------
READS SATISFIED-OUTPUT BUFF     0.00     0.00     N/C     0.00
READS SATISFIED-OUTP.BUF(%)      N/C
READS SATISFIED-ACTIVE LOG      0.00     0.00     N/C     0.00
READS SATISFIED-ACTV.LOG(%)      N/C
READS SATISFIED-ARCHIVE LOG     0.00     0.00     N/C     0.00
READS SATISFIED-ARCH.LOG(%)      N/C

TAPE VOLUME CONTENTION WAIT     0.00     0.00     N/C     0.00
READ DELAYED-UNAVAIL.RESOUR     0.00     0.00     N/C     0.00
ARCHIVE LOG READ ALLOCATION     0.00     0.00     N/C     0.00
ARCHIVE LOG WRITE ALLOCAT.      0.00     0.00     N/C     0.00
CONTR.INTERV.OFFLOADED-ARCH     0.00     0.00     N/C     0.00
LOOK-AHEAD MOUNT ATTEMPTED      0.00     0.00     N/C     0.00
LOOK-AHEAD MOUNT SUCCESSFUL     0.00     0.00     N/C     0.00

UNAVAILABLE OUTPUT LOG BUFF     0.00     0.00     N/C     0.00
OUTPUT LOG BUFFER PAGED IN      0.00     0.00     N/C     0.00

LOG RECORDS CREATED  A        969.5K  3229.17     N/C     6.16
LOG CI CREATED  B           21483.52    71.61     N/C     0.14
LOG WRITE I/O REQ (COPY1&2)   125.3K   417.45     N/C     0.80
LOG CI WRITTEN (COPY1&2)      128.5K   428.08     N/C     0.82
LOG RATE FOR 1 LOG (MB/Sec)      N/A     0.84    N/A    N/A
LOG WRITE SUSPENDED           314.6K  1047.76     N/C     2.00
```

*Figure 69. Log statistics in the DB2 PM statistics report*

- **Choose fast devices for log data sets:** The devices assigned to the active log data sets must be fast ones. Because of its very high sequential performance, ESS is particularly recommended in environments in which the write activity is high to avoid logging bottlenecks.
- **Avoid device contention:** Place the copy of the bootstrap data set and, if using dual active logging, the copy of the active log data sets, on volumes that are accessible on a path different than that of their primary counterparts.
- **Preformat new active log data sets:** Whenever you allocate new active log data sets, preformat them using the DSNJLOGF utility described in Part 3 of *DB2 Utility Guide and Reference*. This action avoids the overhead of preformatting the log, which normally occurs at unpredictable times.

## Log reads

During a rollback, restart, and database recovery, the performance impact of log reads is evident. DB2 must read from the log and apply changes to the data on disk. Every process that requests a log read has an input buffer dedicated to that process. DB2 searches for log records in the following order:

1. Output buffer
2. Active log data set
3. Archive log data set

If the log records are in the output buffer, DB2 reads the records directly from that buffer. If the log records are in the active or archive log, DB2 moves those log records into the input buffer used by the reading process (such as a recovery job or a rollback).

It is always fastest for DB2 to read the log records from the active log rather than the archive log. Access to archived information can be delayed for a considerable length of time if a unit is unavailable or if a volume mount is required (for example, a tape mount).

***Recommendations:***

- **Archive to disk:** If the archive log data set resides on disk, it can be shared by many log readers. In contrast, an archive on tape cannot be shared among log readers. Although it is always best to avoid reading archives altogether, if a process must read the archive, that process is serialized with anyone else who must read the archive tape volume. For example, every rollback that accesses the archive log must wait for any previous rollback work that accesses the same archive tape volume to complete.
- **Avoid device contention on the log data sets:** Place your active log data sets on different volumes and I/O paths to avoid I/O contention in periods of high concurrent log read activity.

  When there are multiple concurrent readers of the active log, DB2 can ease contention by assigning some readers to a second copy of the log. Therefore, for performance and error recovery, use dual logging and place the active log data sets on a number of different volumes and I/O paths. Whenever possible, put data sets within a copy or within different copies on different volumes and I/O paths. Ensure that no data sets for the first copy of the log are on the same volume as data sets for the second copy of the log.

## Log capacity

The capacity you specify for the active log affects DB2 performance significantly. If you specify a capacity that is too small, DB2 might need to access data in the archive log during rollback, restart, and recovery. Accessing an archive takes a considerable amount of time.

The following subsystem parameters affect the capacity of the active log. In each case, increasing the value you specify for the parameter increases the capacity of the active log. See Part 2 of *DB2 Installation Guide* for more information on updating the active log parameters. The parameters are:

- The NUMBER OF LOGS field on installation panel DSNTIPL controls the number of active log data sets you create.
- The ARCHIVE LOG FREQ field on installation panel DSNTIPL is where you provide an estimate of how often active log data sets are copied to the archive log.
- The UPDATE RATE field on installation panel DSNTIPL is where you provide an estimate of how many database changes (inserts, update, and deletes) you expect per hour.

  The DB2 installation CLIST uses UPDATE RATE and ARCHIVE LOG FREQ to calculate the data set size of each active log data set.
- The CHECKPOINT FREQ field on installation panel DSNTIPN specifies the number of log records that DB2 writes between checkpoints or the number of minutes between checkpoints.

This section goes into more detail on the relationships among these parameters and their effects on operations and performance.

### Total capacity and the number of logs

Determining the configuration of your active log data sets is challenging. That is, you need to have sufficient capacity in the active log to avoid reading the archives, and you need to consider how that total capacity should be divided. Having too many or too few active log data sets has ramifications. This information is summarized in Table 83.

*Table 83. The effects of installation options on log data sets. You can modify the size of the data sets in installation job DSNTIJIN*

| Value for ARCHIVE LOG FREQ | Value for NUMBER OF LOGS | Result |
|---|---|---|
| Low | High | Many small data sets. Can cause operational problems when archiving to tape. Checkpoints occur too frequently. |
| High | Low | Few large data sets. Can result in a shortage of active log data sets. |

***Choosing a checkpoint frequency:*** At least one checkpoint is taken each time DB2 switches to a new active log data set. If the data sets are too small, checkpoints occur too frequently. As a result, database writes are not efficient. As a rule of thumb, provide enough active log space for at least 10 checkpoint intervals.

For estimation purposes, assume that a single checkpoint writes 24 KB (or 6 control intervals) of data to the log. A checkpoint interval is defined by the number you specify for checkpoint frequency (the CHECKPOINT FREQ subsystem parameter). You can specify the interval in terms of the number of log records that are written between checkpoints or the number of minutes between checkpoints. Avoid taking more than one checkpoint per minute by raising the CHECKPOINT FREQ value so that the checkpoint interval becomes at least one minute during peak periods. You can change CHECKPOINT FREQ dynamically with the SET LOG or SET SYSPARM command.

***Tips on setting the size of active log data sets:*** You can modify installation job DSNTIJIN to change the size of your active log data set. Some things to consider:

- When you calculate the size of the active log data set, identify the longest unit of work in your application programs. For example, if a batch application program commits only once every 20 minutes, the active log data set should be twice as large as the update information produced during this period by all of the application programs that are running.

  Allow time for possible operator interventions, I/O errors, and tape drive shortages if off-loading to tape. DB2 supports up to 20 tape volumes for a single archive log data set. If your archive log data sets are under the control of DFSMShsm, also consider the Hierarchical Storage Manager recall time, if the data set has been migrated by Hierarchical Storage Manager.

  For more information on determining and setting the size of your active log data sets, refer to *DB2 Installation Guide*.
- When archiving to disk, set the primary space quantity and block size for the archive log data set so that you can offload the active log data set without forcing the use of secondary extents in the archive log data set. This action avoids space abends when writing the archive log data set.
- Make the number of records for the active log be divisible by the blocking factor of the archive log (disk or tape).

  DB2 always writes complete blocks when it creates the archive log copy of the active log data set. If you make the archive log blocking factor evenly divisible into the number of active log records, DB2 does not have to pad the archive log data set with nulls to fill the block. This action can prevent REPRO errors if you should ever have to REPRO the archive log back into the active log data set, such as during disaster recovery.

To determine the blocking factor of the archive log, divide the value specified on the BLOCK SIZE field of installation panel DSNTIPA by 4096 (that is, BLOCK SIZE / 4096). Then modify the DSNTIJIN installation job so that the number of records in the DEFINE CLUSTER field for the active log data set is a multiple of the blocking factor.

- If you offload to tape, consider adjusting the size of each of your active log data sets to contain the same amount of space as can be stored on a nearly full tape volume. This minimizes tape handling and volume mounts and maximizes the use of the tape resource.

  If you change the size of your active log data set to fit on one tape volume, remember that the bootstrap data set is copied to the tape volume along with the copy of the active log data set. Therefore, decrease the size of your active log data set to offset the space required on the archive tape for the bootstrap data set.

## Controlling the amount of log data

Certain processes cause a large amount of information to be logged, requiring a large amount of log space.

### Utilities

The utility operations REORG and LOAD LOG(YES) cause all reorganized or loaded data to be logged. For example, if a table space contains 200 million rows of data, this data, along with control information, is logged when this table space is the object of a REORG utility job. If you use REORG with the DELETE option to eliminate old data in a table and run CHECK DATA to delete rows that are no longer valid in dependent tables, you can use LOG(NO) to control log volume.

**Recommendation:** When populating a table with many records or reorganizing table spaces or indexes, specify LOG(NO) and take an inline copy or take a full image copy immediately after the LOAD or REORG.

Specify LOG(YES) when adding less than 1% of the total table space. This creates additional logging, but eliminates the need for a full image copy.

### SQL

The amount of logging performed for applications depends on how much data is changed. Certain SQL statements are quite powerful, making it easy to modify a large amount of data with a single statement. These statements include:

- INSERT with a fullselect
- Mass deletes and mass updates (except for deleting all rows for a table in a segmented table space)
- Data definition statements log an entire database descriptor for which the change was made. For very large DBDs, this can be a significant amount of logging.
- Modification to a row that contains a LOB column defined as LOG YES.

For nonsegmented table spaces, each of these statements results in the logging of all database data that change. For example, if a table contains 200 million rows of data, that data and control information are logged if all of the rows are deleted in a table using the SQL DELETE statement. No intermediate commit points are taken during this operation.

For segmented table spaces, a mass delete results in the logging of the data of the deleted records when any of the following conditions are true:

- The table is the parent table of a referential constraint.
- The table is defined as DATA CAPTURE(CHANGES), which causes additional information to be logged for certain SQL operations.
- A delete trigger is defined on the table.

***Recommendations:***

- For **mass delete** operations, consider using segmented table spaces. If segmented table spaces are not an option, create one table per table space and use LOAD REPLACE with no rows in the input data set to empty the entire table space.
- For **inserting a large amount of data**, instead of using an SQL INSERT statement, use the LOAD utility with LOG(NO) and take an inline copy.
- For **updates**, consider your workload when defining a table's columns. The amount of data that is logged for update depends on whether the row contains all fixed-length columns or not. For fixed-length rows, changes are logged only from the beginning of the first updated column to the end of the last updated column.

  For varying-length rows, data is logged from the first changed byte to the end of the last updated column. (A varying-length row contains one or more varying-length columns.)

  To determine your workload type, read-intensive or update-intensive, check the log data rate. Use the formula in "Calculating average log record size" on page 606 to determine the average log size and divide that by 60 to get the average number of log bytes written per second.
  - If you log less than 1 MB per second, the workload is read-intensive.
  - If you log more than 1 MB per second, it is an update-intensive workload.

  Table 84 summarizes the recommendations for the type of row and type of workload you run.

*Table 84. Recommendations for database design to reduce log quantities*

| | **Workload** | |
|---|---|---|
| **Row type** | **Read-intensive** | **Update-intensive** |
| Fixed-length | | Keep frequently updated columns close to each other. |
| Varying-length | Keep varying-length columns at the end of the row to improve read performance. | Keep all frequently updated columns near the end of the row. However, if only fixed-length columns will be updated, keep those columns close to each other at the beginning of the row. |

- If you have many data definition statements (CREATE, ALTER, DROP) for a single database, issue them within a single unit of work to avoid logging the changed DBD for each data definition statement. However, be aware that the DBD is locked until the COMMIT is issued.
- Use LOG NO for any LOBs that require frequent updating and for which the tradeoff of nonrecoverability of LOB data from the log is acceptable. (You can still use the RECOVER utility on LOB table spaces to recover control information that ensures physical consistency of the LOB table space.)

Because LOB table spaces defined as LOG NO are nonrecoverable from the DB2 log, make a recovery plan for that data. For example, if you run batch updates, be sure to take an image copy after the updates are complete.

### Calculating average log record size

One way to determine how much log volume you need is to calculate the average size of log records written. To do this, you need values from the statistics report shown in Figure 69 on page 601: the LOG RECORDS CREATED counter ( A ) and the number of control intervals created in the active log counter ( B ). Use the following formula:

```
B  × 4 KB / A  = avg size of log record
```

## Improving disk utilization: space and device utilization

To use disk space more efficiently, you can:

- Change your allocation of data sets to keep data sets within primary allocations. To understand how DB2 extends data sets, see "Allocating and extending data sets".
- Manage them with the Hierarchical Storage Management functional component (DFSMShsm) of DFSMS, as described in "Managing your DB2 data sets with DFSMShsm™" on page 37.
- Compress your data, as described in "Compressing your data".
- Choose a page size that gives you good disk use and I/O performance characteristics, as described in "Choosing a page size" on page 43.

To manage the use of disk, you can use RMF to monitor how your devices are used. Watch for usage rates that are higher than 30% to 35%, and for disk devices with high activity rates. Log devices could have usage rates of up to 50% without having serious performance problems.

## Allocating and extending data sets

Primary and secondary allocation sizes are the main factors that affect the amount of disk space that DB2 uses.

In general, the primary allocation must be large enough to handle the storage needs that you anticipate. The secondary allocation must be large enough for your applications to continue operating until the data set is reorganized.

If the secondary allocation space is too small, the data set might have to be extended more times to satisfy those activities that need a large space.

IFCID 0258 allows you to monitor data set extension activities by providing information, such as the primary allocation quantity, maximum data set size, high allocated space before and after extension activity, number of extents before and after the extend, maximum volumes of a VSAM data set, and number of volumes before and after the extend. Access IFCID 0258 in Statistics Class 3 (SC03) through an IFI READA request.

See "Chapter 4. Creating storage groups and managing DB2 data sets" on page 31 for more information about extending data sets.

## Compressing your data

You can compress data in a table space or partition by specifying COMPRESS YES on CREATE TABLESPACE or ALTER TABLESPACE and then running the LOAD or

REORG utility. When you compress data, bit strings that occur frequently are replaced by shorter strings. Information about the mapping of bit strings to their replacements is stored in a *compression dictionary*. Computer processing is required to compress data before it is stored and to decompress the data when it is retrieved from storage. In many cases, using the COMPRESS clause can significantly reduce the amount of disk space needed to store data, but the compression ratio you achieve depends on the characteristics of your data.

With compressed data, you might see some of the following performance benefits, depending on the SQL work load and the amount of compression:

- Higher buffer pool hit ratios
- Fewer I/Os
- Fewer getpage operations

As described under "Determining the effectiveness of compression" on page 609, you can use the DSN1COMP utility to determine how well your data will compress. Data in a LOB table space or a table space that is defined in a TEMP database (a table space for declared temporary tables) cannot be compressed.

## Deciding whether to compress

Consider these points before compressing data:

- Data row size

  DB2 compresses the data of one record at a time. (The prefix of the record is not compressed.) As row lengths become shorter, compression yields diminishing returns because 8 bytes of overhead are required to store each record in a data page. On the other hand, when row lengths are very long, compression of the data portion of the row may yield little or no reduction in data set size because DB2 rows cannot span data pages. In the case of very long rows, using a larger page size can enhance the benefits of compression, especially if the data is accessed primarily in a sequential mode.

  If compressing the record produces a result that is no shorter than the original, DB2 does not compress the record.

- Table space size

  Compression can work very well for large table spaces. With small table spaces, the size of the compression dictionary (8 KB to 64 KB) can offset the space savings that compression provides.

- Processing costs

  Compressing data can result in a higher processor cost, depending on the SQL work load. However, if you use IBM's synchronous data compression hardware, processor time is significantly less than if you use just the DB2-provided software simulation or an edit or field procedure to compress the data.

  Decompressing a row of data costs significantly less than compressing that same row. This rule applies regardless of whether the compression uses the synchronous data compression hardware or the software simulation that is built into DB2.

  The data access path that DB2 uses affects the processor cost for data compression. In general, the relative overhead of compression is higher for table space scans and is less costlier for index access.

- I/O costs

  When rows are accessed sequentially, fewer I/Os might be required to access data that is stored in a compressed table space. However, there is a tradeoff between reduced I/O resource consumption and the extra processor cost for decoding the data.

  - If random I/O is necessary to access the data, the number of I/Os will not decrease significantly, unless the associated buffer pool is larger than the table and the other applications require little concurrent buffer pool usage.
  - Some types of data compress better than others. Data that contains hexadecimal characters or strings that occur with high frequency compresses quite well, while data that contains random byte frequencies might not compress at all. For example, textual and decimal data tends to compress well because certain byte strings occur frequently.
- Data patterns

  The frequency of patterns in the data determines the compression savings. Data with many repeated strings (such as state and city names or numbers with sequences of zeros) results in good compression savings.
- Table space design

  Each table space or partition that contains compressed data has a compression dictionary, which is built by using the LOAD utility with the REPLACE or RESUME NO options or the REORG TABLESPACE utility. The dictionary contains a fixed number of entries, usually 4096, and resides with the data. The dictionary content is based on the data at the time it was built, and does not change unless the dictionary is rebuilt or recovered, or compression is disabled with ALTER TABLESPACE.

  If you use LOAD to build the compression dictionary, the first *n* rows loaded in the table space determine the contents of the dictionary. The value of *n* is determined by how much your data can be compressed.

  If you have a table space with more than one table and the data used to build the dictionary comes from only one or a few of those tables, the data compression might not be optimal for the remaining tables. Therefore, put a table you want to compress into a table space by itself, or into a table space that only contains tables with similar kinds of data.

  REORG uses a sampling technique to build the dictionary. This technique uses the first *n* rows from the table space and then continues to sample rows for the remainder of the UNLOAD phase. In most cases, this sampling technique produces a better dictionary than does LOAD, and using REORG might produce better results for table spaces that contain tables with dissimilar kinds of data.

  For more information about using LOAD or REORG to create a compression dictionary, see Part 2 of *DB2 Utility Guide and Reference*.
- Existing exit routines

  An exit routine is executed before compressing or after decompressing, so you can use DB2 data compression with your existing exit routines. However, do not use DB2 data compression in conjunction with DSN8HUFF. (DSN8HUFF is a sample edit routine that compresses data using the Huffman algorithm, which is provided with DB2). This adds little additional compression at the cost of significant extra CPU processing.
- Logging effects

  If a data row is compressed, all data that is logged because of SQL changes to that data is compressed. Thus, you can expect less logging for insertions and deletions; the amount of logging for updates varies. Applications that are sensitive to log-related resources can experience some benefit with compressed data.

  External routines that read the DB2 log cannot interpret compressed data without access to the compression dictionary that was in effect when the data was compressed. However, using IFCID 306, you can cause DB2 to write log records of compressed data in decompressed format. You can retrieve those decompressed records by using the IFI function READS.

- Distributed data

  DB2 decompresses data before transmitting it to VTAM.

### Tuning recommendation

There are some cases where using compressed data results in an increase in the number of getpages, lock requests, and synchronous read I/Os. Sometimes, updated compressed rows cannot fit in the home page, and they must be stored in the overflow page. This can cause additional getpage and lock requests. If a page contains compressed fixed-length rows with no free space, an updated row probably has to be stored in the overflow page.

To avoid the potential problem of more getpage and lock requests, add more free space within the page. Start with 10 percent additional free space and adjust further, as needed. If, for example, 10 percent free space was used without compression, then start with 20 percent free space with compression for most cases. This recommendation is especially important for data that is heavily updated.

### Determining the effectiveness of compression

Before compressing data, you can use the DSN1COMP stand-alone utility to estimate how well it will compress. After data is compressed, use compression reports and catalog statistics to determine how effectively it was compressed.

- DSN1COMP

  Use stand-alone utility DSN1COMP to find out how much space it will save and how much processing it will require to compress your data. Run DSN1COMP on a data set that contains a table space, a table space partition, or an image copy. DSN1COMP generates a report of compression statistics but does not compress the data. For instructions on using DSN1COMP, see Part 3 of *DB2 Utility Guide and Reference*.

- Compression reports

  Examine the compression reports after you use REORG or LOAD to build the compression dictionary and compress the data. Both utilities issue a report message (DSNU234I or DSNU244I). This report message gives information about how well the data is compressed and how much space is saved. (REORG with the KEEPDICTIONARY option does not produce the report.)

- Catalog statistics

  In addition to the compression reports, these columns in the catalog tables contain information about data compression:

  – PAGESAVE column of the SYSIBM.SYSTABLEPART tells you the percentage of pages that are saved by compressing the data.

  – PCTROWCOMP columns of SYSIBM.SYSTABLES and SYSIBM.SYSTABSTATS tells you the percentage of the rows that were compressed in the table or partition the last time RUNSTATS was run. Use the RUNSTATS utility to update these catalog columns.

## Improving main storage utilization

This section provides specific information for both real and virtual storage tuning. With DB2, the amount of real storage often needs to be close to the amount of virtual storage. For a general overview of some factors relating to virtual storage planning, see Part 2 of *DB2 Installation Guide*.

Use the techniques listed below to reduce your use of virtual storage.

***Minimize storage needed for locks:*** You can save main storage by using the LOCKSIZE TABLESPACE option on the CREATE TABLESPACE statements for large tables, which affects concurrency. This option is most practical when concurrent read activity without a write intent, or a single write process, is used.

You can use LOCKSIZE PAGE or LOCKSIZE ROW more efficiently when you commit your data more frequently or when you use cursor stability with CURRENTDATA NO. For more information on specifying LOCKSIZE TABLESPACE, see “Monitoring of DB2 locking” on page 700.

***Reduce the number of open data sets:*** You can reduce the number of open data sets by:

- Including multiple tables in segmented table spaces
- Using fewer indexes
- Reducing the value you use for DSMAX

***Reduce the unnecessary use of DB2 sort:*** DB2 sort uses buffer pool 0 and database DSNDB07, which holds the temporary work files. However, to obtain more specific information for tuning, you can assign the temporary work file table spaces in DSNDB07 to another buffer pool. Using DB2 sort increases the load on the processor, on virtual and real storage, and on I/O devices. Hints for reducing the need to sort are described in “Overview of index access” on page 806.

***Provide for type 2 inactive threads:*** As described in “Using type 2 inactive threads” on page 626, distributed threads that are allowed to go inactive use less storage than active threads. Type 2 inactive threads take even less storage than type 1 inactive threads. Type 1 inactive threads are around 70 KB of storage in the *ssnm*DBM1 address space per thread. Type 2 inactive threads, on the other hand, are only about 8 KB per thread, and that storage is in the DDF address space (*ssnm*DIST) rather than in *ssnm*DBM1.

***Ensure ECSA size is adequate:*** The extended common service area (ECSA) is a system area that DB2 shares with other programs. Shortage of ECSA at the system level leads to use of the common service area.

DB2 places some load modules and data into the common service area. These modules require primary addressability to any address space, including the application’s address space. Some control blocks are obtained from common storage and require global addressability. For more information, see Part 2 of *DB2 Installation Guide*.

***Ensure EDM pool space is being used efficiently:*** Monitor your use of EDM pool storage using DB2 statistics and see “Tips for managing EDM pool storage” on page 573, which includes information about using data spaces for EDM pool storage used for dynamic statement caching.

***Use less buffer pool storage:*** Using fewer and smaller virtual buffer pools reduces the amount of central storage space DB2 requires. Virtual buffer pool size can also affect the number of I/O operations performed; the smaller the virtual buffer pool, the more I/O operations needed. Also, some SQL operations, such as joins, can create a result row that will not fit on a 4 KB page. For information about this, see “Make buffer pools large enough for the workload” on page 540.

See “Buffer pools and data spaces” on page 552 for information about putting virtual buffer pools in data spaces, another way to reduce storage in DB2’s address space.

***Control maximum number of LE tokens:*** When a function is executed and needs to access storage used by LE/370, it obtains an LE token from the pool. LE/370 provides a common runtime environment for programming languages. A token is taken each time one of the following functions is executed:

- Log functions (LOG, LN, LOG10)
- Trigonometry functions (ACOS, ASIN, ATAN, ATANH, ATAN2, COS, COSH, SIN, SINH, TAN, and TANH)
- EXP
- POWER
- RAND
- ADD_MONTHS
- LAST_DAY
- NEXT_DAY
- ROUND_TIMESTAMP
- TRUNC_TIMESTAMP
- LOWER
- TRANSLATE
- UPPER

Upon completion of the call to LE, the token is returned to the pool. The MAXIMUM LE TOKENS (LEMAX) field on the DSNTIP7 panel controls the maximum number of LE tokens that are active at any time. The LEMAX default value is 20 with a range of 0 to 50. If the value is zero, no tokens are available. If a large number of functions are executing at the same time, all the token may be used. Thus, if a statement needs a token and none is available, the statement is queued. If the statistics trace QLENTRDY is very large, indicating a delay for an application because an LE token is not immediately available, the LEMAX may be too small. If the statistics trace QLETIMEW for cumulative time spent is very large, the LEMAX may be too small. Increase the number of tokens for the MAXIMUM LE TOKENS field on the DSNTIP7 panel. For more information on DSNTIP7, see Part 2 of *DB2 Installation Guide*.

## Performance and the storage hierarchy

To meet the diverse needs of application data, a range of storage options is available, each with different access speeds, capacities, and costs per megabyte. This broad selection of storage alternatives supports requirements for enhanced performance and expanded online storage options, providing more options in terms of performance and price.

The levels in the DB2 storage hierarchy include central storage and expanded storage, storage controller cache, disk, and auxiliary storage.

## Real storage

Real storage refers to the processor storage where program instructions reside while they are executing. Data in DB2's virtual buffer pools resides in virtual storage, which is backed by real, expanded, and auxiliary storage. The maximum amount of real storage that one DB2 subsystem can use is about 2 GB.

## Expanded storage

Expanded storage is optional high-speed processor storage. Data is moved in 4 KB blocks between central storage and expanded storage. Data cannot be transferred to or from expanded storage without passing through central storage.

If your DB2 subsystem is on a processor that has the Fast Sync data mover facility (such as an S/390 G5/G6 enterprise server) or that has the Asynchronous Data Mover hardware feature installed, DB2 can use up to 8 GB of expanded storage by creating hiperpools. For more information on how DB2 uses hiperpools, see “Buffer pools and hiperpools” on page 550.

## Storage controller cache

DB2 can take advantage of storage controller cache. To understand how DB2 can use storage controller cache, you need to understand how that cache storage fits into the storage hierarchy. DB2’s primary “cache” is the central storage and expanded storage in the processor used for such things as virtual buffer pools, the EDM pool and the sort pool.

DB2’s large capacity for buffers in processor storage and its write avoidance and sequential access techniques allow applications to avoid a substantial amount of read and write I/O, combining single accesses into sequential access, so that the disk devices are used more effectively.

The storage controller cache acts as a secondary buffer. It is not useful to store the same data in processor storage and the storage controller cache. To be useful, the storage controller cache must be significantly larger than the buffers in real storage, store different data, or provide another performance advantage.

In addition, using the cache enables other performance and availability enhancements, such as DASD Fast Write, concurrent copy, and dual copy. You can use DFSMS to provide dynamic management of the cache storage.

### The amount of storage controller cache

The amount of cache to use for DB2 depends primarily on the relative importance of price and performance. It is not often effective to have large memory resources for both DB2 buffers and storage controller cache. If you decide to concentrate on the storage controller cache for performance gains, then use the maximum available cache size. If the cache is substantially larger than the DB2 buffer pools, DB2 can make effective use of the cache to reduce I/O times for random I/O.

What constitutes a large cache is based on your configuration. Table 85 lists some configurations and their recommended large cache sizes:

*Table 85. Sample configurations and cache sizes*

| Configuration | ″Large″ cache size |
|---|---|
| RAMAC 2 of 180 GB | 1 GB |
| RVA T82 of 420 GB | 2 GB |
| RSA 3 of 1258 GB | 4 GB |
| ESS of 8 GB | 16 GB |

For sequential I/O, the improvement the cache provides is generally small. However, DB2 data compression and parallel I/O streams can contribute to faster I/O times. Compressing data reduces the amount of data that is sent across the channel, through the controller, and onto disk. Compression also allows you to reduce buffer pool size without reducing buffer pool hit ratios.

## Sequential cache installation option

DB2 provides the option to use or bypass the cache for sequential prefetch. On panel DSNTIPE, you can specify whether to use the sequential mode to read cached data from a 3990 Model or Model 6, ESS, or RVA cache. If you specify SEQ, DB2 sequential prefetch (including sequential detection) uses the cache. If you specify BYPASS, which is the default, DB2 sequential prefetch bypasses the cache. List prefetch always bypasses the cache.

***Recommendation:*** If you have current disk devices with good cache sizes (greater than 1GB), specify SEQ for the cache option on panel DSNTIPE, especially if the units are ESS or RVA. This option can improve performance because data can be transferred between disk and cache by the cylinder rather than by the track.

***Sort work files:*** Sort work files can have a large number of concurrent processes that can overload a storage controller with a small cache and thereby degrade system performance. For instance, one large sort could use 100 sequential files, needing 60 MB of storage. Unless the cache sizes are large, you might need to specify BYPASS or use DFSMS controls to prevent the use of the cache during sort processing. Separate units for sort work can give better performance.

## Utility cache option

If you are using storage controller caching, and you have the nonpartitioning indexes on RAMAC family disks, consider specifying YES on the UTILITY CACHE OPTION field of installation panel DSNTIPE. This allows DB2 to use sequential prestaging when reading data from disk 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.

## Parallel Access Volumes (PAV)

The Parallel Access Volumes (PAV) feature allows multiple concurrent I/Os on a given device when the I/O requests originate from the same system. PAVs make it possible to store multiple partitions on the same volume with almost no loss of performance. In older disk subsystems, if more than one partition is placed on the same volume (intentionally or otherwise), attempts to read the partitions result in contention, which shows up as I/O subsystem queue (IOSQ) time. Without PAVs, poor placement of a single data set can almost double the elapsed time of a parallel query.

## Multiple Allegiance

The Multiple Allegiance feature allows multiple active concurrent I/Os on a given device when the I/O requests originate from different systems. PAVs and multiple allegiance dramatically improve I/O performance for parallel work on the same volume by nearly eliminating IOSQ or PEND time and drastically lowering elapsed time for transactions and queries.

## Fast Write

The Fast Write function can be very effective for synchronous writes. It is recommended especially for use with the DB2 log, improving response times for the log writes that occur at the end of each transaction. For example, for dual logging, response times for the four log writes that occur at commit can be reduced from approximately 50 milliseconds total to approximately 10 milliseconds. In addition, the shorter lock duration required for logging pages of data can provide improved concurrency. Storing adequate amounts of log data on disk is crucial for restart and recovery performance.

## MVS performance options for DB2

You can set MVS performance options for DB2 in two ways:

- Using system resources manager (SRM). This is called “compatibility mode”
- Using goal mode

  In **goal mode**, MVS’s workload manager controls the dispatching priority based on goals you supply. Workload manager raises or lowers the priority as needed to meet the specified goal. A major objective of goal mode is to remove the need to fine tune the exact priorities of every piece of work in the system and to focus instead on business objectives.

  There are three kinds of goals: response-time, velocity, and discretionary. Response times are appropriate goals for “end user” applications, such as QMF users running under the TSO address space goals, or users of CICS using the CICS work load goals. You can also set response time goals for distributed users, as described in “Using Workload Manager to set performance objectives” on page 629. For more information about setting response time goals for users, see *OS/390 MVS Planning: Workload Management*.

For DB2 address spaces, **velocity goals** are more appropriate. A small amount of the work done in DB2 is counted toward this velocity goal (most of it applies to the end user goal described above). Velocity goals indicate how quickly you want your work to be processed.

This section describes ways to set DB2 address space performance options:

- “Using SRM (compatibility mode)”
- “Determining MVS workload management velocity goals” on page 616
- “How DB2 assigns I/O priorities” on page 618

## Using SRM (compatibility mode)

You can run in compatibility mode with few or no changes to existing SRM values.

### Setting address space priority

Review the following SRM options when installing or tuning a DB2 subsystem (see also *DB2 Installation Guide*). Be aware that there are special considerations for where you place the address space for the distributed data facility. Generally, set MVS processor dispatching priorities in the following order, from highest to lowest priority:

| 1. VTAM and TCP/IP address spaces
2. IRLM address space (IRLMPROC)

   **Attention:** It is extremely important that IRLM’s priority be higher than DB2’s. Serious performance problems can occur if it is not.
3. IMS control address space or CICS terminal owning region
| 4. DB2 system services address space (*ssnm*MSTR), DB2 database services
| address space (*ssnm*DBM1), distributed data facility address space
| (*ssnm*DIST), and WLM-established stored procedures address spaces

| The DB2 system services and database services address spaces appear near
| the top of the list because, though work done under DB2 is usually a small part
| of the total, delaying it can delay other users. For example, writes to the log
| might become a bottleneck if not performed with high priority.

MVS considers the distributed data facility address space and WLM-established stored procedure addresses spaces to be service address spaces. As such, to enable new work to be scheduled in them, they need the same priority as the DB2 system services and database services address spaces. For the DDF address space, after the work is classified into an enclave, priorities or goals can be set for the work.

For the WLM-established address spaces, when the work is started, it runs at the same priority of the stored procedure caller (IMS or CICS, for example).

5. Distributed work (SUBSYS=DDF)

   Ensure that you create the SUBSYS=DDF service class definitions. Otherwise, the distributed work loads will default to the priority of the DDF address space (*ssnm*DIST), which will be too high.
6. DB2-established stored procedures address space (*ssnm*SPAS)

   Because stored procedures that run in *ssnm*SPAS run at the priority of *ssnm*SPAS, set the priority of *ssnm*SPAS similarly to that of the calling application.
7. CICS application owning regions
8. IMS dependent regions or TSO address spaces

## I/O scheduling priority

DB2 can schedule read and write I/O's according to an application's address space or enclave as described in "How DB2 assigns I/O priorities" on page 618. To enable this feature, you must do both of the following:

- Enable MVS I/O priority scheduling by specifying IOQ=PRTY in the IEAIPS*xx* member of SYS1.PARMLIB.
- Use the IOP parameter to set the I/O priority for the address space of a performance group. The IOP parameter is in the IEAIPSxx member of SYS1.PARMLIB.

If you specify IOQ=PRTY, it is critical that you specify the proper IOP value for each address space. If IOQ=PRTY is specified and the IOP parameter is not set for an address space, the I/O scheduling priority for that address space defaults to the address space's processor scheduling priority; in other words, the IOP value defaults to the dispatching priority (DP) value.

If you do not specify values for the IOP parameter, CICS and IMS regions might have lower I/O scheduling priority than DB2's *ssnm*DBM1 address space. An I/O scheduling priority lower than *ssnm*DBM1's I/O scheduling priority could result in inconsistent I/O response time for transaction applications.

For more information on the IOP and IOQ parameters, see *OS/390 MVS Initialization and Tuning Reference*.

**Recommendations:**

- To improve response time for transaction processing, set the CICS- and IMS-dependent IOP values higher than DB2's *ssnm*DBM1 address space. To favor transaction processing over query users, set the IOP values for CICS and IMS MPP regions higher than those for TSO and batch users.
- Ensure that *ssnm*DBM1 has a higher priority than TSO and batch to help ensure that deferred write I/Os are scheduled before prefetch read I/Os, thereby preventing a shortage of available buffers.

### Storage isolation

DB2 allows page faults to occur without significantly affecting overall system performance. Therefore, DB2 storage does not need to be protected with the SRM storage isolation. However, if other subsystems use SRM storage isolation, provide it also for the DB2 and IRLM address spaces.

### Workload control

Performance groups and performance-group periods can be used effectively to prioritize the TSO, batch, QMF, and distributed work loads. This way, long queries can be dispatched with lower priority and can be swapped-out, allowing short queries to complete. However, this approach causes DB2 resources used by these low priority queries to be held for more time. Watch for lock contention and lock suspensions caused by swapped-out users; perhaps your work load can be managed to avoid resource usage swap-outs.

## Determining MVS workload management velocity goals

To determine velocity goals, you can start by determining an address space's velocity while you are running your systems in compatibility mode. You can define a report performance group for the address space, or group of address spaces, in which you are interested, and review the RMF Monitor I workload activity report, which shows the execution velocity of that report performance group in compatibility mode. Gather this information during peak work times.

As a starting point, you can then define a service goal with the same value for the work defined in a service class.

This section tells you how to set velocity goals in two situations: 1) an interim situation in which you have not yet determined response time goals for applications or in which you do not have the prerequisite software to do so, and 2) you have determined response time goals and are ready to fully implement MVS WLM goal mode.

### Recommendations for an interim situation

If your installation is not yet managing CICS, IMS, or DDF transactions according to MVS WLM response time goals, or if you have not yet gotten the required release levels to do so, consider the following service class definitions.

- The MVS workload manager default service class for started tasks (SYSSTC) for the following address spaces:

  VTAM and TCP/IP address spaces

  IRLM address space (IRLMPROC)

  **Attention:** The VTAM, TCP/IP, and IRLM address spaces must always have a higher dispatching priority than all of the DBMS address spaces, their attached address spaces, and their subordinate address spaces. Do not allow WLM to reduce the priority of VTAM, TCP/IP, or IRLM to or below that of the other DBMS address spaces.

- A service class with a medium to high velocity goal with a name you define, such as PRODCNTL, for the following:

  IMS control address space

  DB2 (all address spaces, except for the DB2-established stored procedures address space).

  Set any work done by distributed tasks (SUBSYS=DDF) in a lower service class than the service class for these DB2 address spaces, which should be in the same service class.

  CICS terminal-owning regions

- A service class with a lower velocity or importance than PRODCNTL with a name you define, such as PRODREGN, for the following:

  IMS-dependent regions

  CICS application-owning regions

  The DB2-established stored procedures address space (*ssnm*SPAS) and any WLM-established stored procedures address spaces
- Set the DB2 distributed data address space (*ssnm*DIST) in the same service class as *ssnm*DBM1.

## Recommendations for full implementation of MVS WLM

If your installation is managing CICS, IMS, or DDF transactions according to MVS WLM response time goals and if you are set up to use WLM-established stored procedures address spaces, use following service classes for velocity:

- The default SYSSTC service class for:

  | VTAM and TCP/IP address spaces

  IRLM address space (IRLMPROC)

  **Attention:** The VTAM, TCP/IP, and IRLM address spaces must always have a higher dispatching priority than all of the DBMS address spaces, their | attached address spaces, and their subordinate address spaces. Do not allow | WLM to reduce the priority of VTAM, TCP/IP, or IRLM to or below that of the | other DBMS address spaces.
- A high velocity goal for a service class whose name you define, such as PRODREGN, for the following:

  DB2 (all address spaces, except for the DB2-established stored procedures address space)

  CICS (all region types)

  IMS (all region types except BMPs)

The velocity goals for CICS and IMS regions are only important during startup or restart. After transactions begin running, WLM ignores the CICS or IMS velocity goals and assigns priorities based on the goals of the transactions that are running in the regions. A high velocity goal is good for ensuring that startups and restarts are performed as quickly as possible.

Similarly, when you set response time goals for DDF threads or for stored procedures in a WLM-established address space, the only work controlled by the DDF or stored procedure velocity goals are the DB2 service tasks (work performed for DB2 that cannot be attributed to a single user). The user work runs under separate goals for the enclave, as described in "Using Workload Manager to set performance objectives" on page 629.

For the DB2-established stored procedures address space, use a velocity goal that reflects the requirements of your distributed work. Depending on what type of distributed work you do, this might be equal to or lower than the goal for PRODREGN.

IMS BMPs can be treated along with other batch jobs or given a velocity goal, depending on what business and functional requirements you have at your site.

## Other considerations

- IRLM must be eligible for the SYSSTC service class. To make IRLM eligible for SYSSTC, do not classify IRLM to one of your own service classes.

- If you need to change a goal, changing the velocity by 2 or 3% is not noticeable. Velocity goals don't translate directly to priority. Higher velocity tends to have higher priority, but this is not always the case.
- WLM in goal mode can assign I/O priority (based on I/O delays) separately from processor priority. In compatibility mode, WLM assigns I/O priority based on what you specify in the IPS PARMLIB member. Goal mode does *not* use the IPS PARMLIB member.

  See "How DB2 assigns I/O priorities" for information about how read and write I/O priorities are determined.
- MVS workload management dynamically manages storage isolation to meet the goals you set.

## How DB2 assigns I/O priorities

DB2 informs MVS about which address space's priority is to be associated with a particular I/O request. Then MVS workload manager handles the management of the request from there, as described earlier in this section. Table 86 and Table 87 describe to which enclave or address space DB2 associates I/O read and write requests.

*Table 86. How read I/O priority is determined*

| Request type | Synchronous reads | Prefetch reads |
|---|---|---|
| Local | Application's address space | Application's address space |
| DDF or Sysplex query parallelism (assistant only) | Enclave priority | Enclave priority |

*Table 87. How write I/O priority is determined*

| Request type | Synchronous writes | Deferred writes |
|---|---|---|
| Local | Application's address space | ssnmDBM1 address space |
| DDF | DDF address space | ssnmDBM1 address priority |

# Chapter 29. Managing DB2 threads

Threads are an important DB2 resource. When you install DB2, you choose a maximum number of active allied and database access threads that can be allocated concurrently. Choosing a good number for this is important to keep applications from queuing and to provide good response time.

When writing an application, you should know when threads are created and terminated and when they can be reused, because thread allocation can be a significant part of the cost in a short transaction.

This chapter provides a general introduction on how DB2 uses threads. It includes the following sections:

- A discussion of how to choose the maximum number of concurrent threads, in “Setting thread limits”
- A description of the steps in creating and terminating an allied thread, in “Allied thread allocation” on page 620
- An explanation of the differences between allied threads and database access threads (DBATs) and a description of how DBATs are created, including how they become active or inactive and how to set performance goals for individual DDF threads, under “Database access threads” on page 624
- Design options for reducing thread allocations and improving performance generally, under “CICS design options” on page 633, “IMS design options” on page 639, and “TSO design options” on page 640

## Setting thread limits

You set the limit of the number of allied and database access threads that can be allocated concurrently using fields MAX USERS and MAX REMOTE ACTIVE on installation panel DSNTIPE. The combined maximum allowed for MAX USERS and MAX REMOTE ACTIVE is 2000.

Set these values to provide good response time without wasting resources, such as virtual and real storage. The value you specify depends upon your machine size, your work load, and other factors. When specifying values for these fields, consider the following:

- Fewer threads than needed under utilize the processor and cause queuing for threads.
- More threads than needed do not improve the response time. They require more real storage for the additional threads and might cause more paging and, hence, performance degradation.

If real storage is the limiting factor, set MAX USERS and MAX REMOTE ACTIVE according to the available storage. For more information on storage, refer to Part 2 of *DB2 Installation Guide*.

***Thread limits for TSO and call attachment:*** For the TSO and call attachment facilities, you limit the number of threads indirectly by choosing values for the MAX TSO CONNECT and MAX BATCH CONNECT fields of installation panel DSNTIPE. These values limit the number of connections to DB2. The number of threads and connections allowed affects the amount of work that DB2 can process.

# Allied thread allocation

This section describes at a high level the steps in allocating an allied thread, and some of the factors related to the performance of those steps. This section does not explain how a database access thread is allocated. For more information on database access threads, see “Database access threads” on page 624.

# Step 1: Thread creation

During thread creation with ACQUIRE(ALLOCATE), the resources needed to execute the application are acquired. During thread creation with ACQUIRE(USE), only the thread is created.

The following list shows the main steps in thread creation.

1. Check the maximum number of threads.

   DB2 checks whether the maximum number of active threads, specified as MAX USERS for local threads or MAX REMOTE ACTIVE for remote threads on the Storage Sizes panel (DSNTIPE) when DB2 was installed, has been exceeded. If it has been exceeded, the request waits. The wait for threads is not traced, but the number of requests queued is provided in the performance trace record with IFCID 0073.
2. Check the plan authorization.

   The authorization ID for an application plan is checked in the SYSPLANAUTH catalog table (IFCID 0015). If this check fails, the table SYSUSERAUTH is checked for the SYSADM special privilege.
3. For an application plan, load the control structures associated with the plan.

   The control block for an application plan is divided into sections. The header and directory contain control information; SQL sections contain SQL statements from the application. A copy of the plan’s control structure is made for each thread executing the plan. Only the header and directory are loaded when the thread is created.
4. Load the descriptors necessary to process the plan.

   Some of the control structures describe the DB2 table spaces, tables, and indexes used by the application. If ACQUIRE(ALLOCATE) is used, all the descriptors referred to in the plan are loaded now. If the plan is bound with ACQUIRE(USE), they are loaded when SQL statements are executed.

## Performance factors in thread creation

The most relevant factors from a system performance point of view are:

***Thread reuse:*** Thread creation is a significant cost for small and medium transactions. When execution of a transaction is terminated, the thread can sometimes be reused by another transaction using the same plan. For more information on thread reuse, see “Providing for thread reuse” on page 623.

***ACQUIRE option of BIND:*** ACQUIRE(ALLOCATE) causes all the resources referred to in the application to be allocated when the thread is created. ACQUIRE(USE) allocates the resources only when an SQL statement is about to be executed. In general, ACQUIRE(USE) is recommended. However, if most of the SQL is used in every execution of the transaction, ACQUIRE(ALLOCATE) is cheaper.

***EDM pool size***: The size of the EDM pool influences the number of I/Os needed to load the control structures necessary to process the plan or package. To avoid a large number of allocation I/Os, the EDM pool must be large enough to contain the structures that are needed. See “Tuning the EDM pool” on page 570 for more information.

## Step 2: Resource allocation

Some of the structures necessary to process the statement are stored in 4 KB pages. If they are not already present, those are read into database buffer pool BP0 and copied from there into the EDM pool. If the plan was bound with ACQUIRE(USE), it acquires resources when the statement is about to execute.

1. Load the control structures necessary to process the SQL section.

   If it is not already in the EDM pool, DB2 loads the control structure’s section corresponding to this SQL statement.
2. Load structures necessary to process statement.

   Load any structures referred to by this SQL statement that are not already in the EDM pool.
3. Allocate and open data sets.

   When the control structure is loaded, DB2 locks the resources used.

### Performance factors in resource allocation

The most important factors are the same as that for thread creation.

## Step 3: SQL statement execution

If the statement resides in a package, the directory and header of the package’s control structure is loaded at the time of the first execution of a statement in the package. The control structure for the package is allocated at statement execution time. This is contrasted with the control structures for plans bound with ACQUIRE(ALLOCATE), which are allocated at thread creation time. The header of the plan’s control structures is allocated at thread creation time regardless of ACQUIRE(ALLOCATE) or ACQUIRE(USE).

When the package is allocated, DB2 checks authorization using the package authorization cache or the SYSPACKAUTH catalog table. DB2 checks to see that the plan owner has execute authority on the package. On the first execution, the information is not in the cache; therefore, the catalog is used. Thereafter, the cache is used. For more information about package authorization caching, see “Caching authorization IDs for best performance” on page 120.

Authorization checking also occurs at statement execution time.

A summary record, produced at the end of the statement (IFCID 0058), contains information about each scan performed. Included in the record is the following information:

- The number of rows updated
- The number of rows processed
- The number of rows deleted
- The number of rows examined
- The number of pages requested through a getpage operation
- The number of rows evaluated during the first stage (stage 1) of processing
- The number of rows evaluated during the second stage (stage 2) of processing
- The number of getpage requests issued to enforce referential constraints
- The number of rows deleted or set null to enforce referential constraints
- The number of rows inserted

### Performance factors in SQL statement execution

From a system performance perspective, the most important factor is the size of the database buffer pool. If the buffer pool is large enough, some index and data pages can remain there and can be accessed again without an additional I/O operation. For more information on buffer pools, see “Chapter 27. Tuning DB2 buffer, EDM, RID, and sort pools” on page 549.

## Step 4: Commit and thread termination

Commit processing can occur many times while a thread is active. For example, an application program running under the control structure of the thread could issue an explicit COMMIT or SYNCPOINT several times during its execution. When the application program or the thread terminates, an implicit COMMIT or SYNCPOINT is issued.

When a COMMIT or SYNCPOINT is issued from an IMS application running with DB2, the two-phase commit process begins if DB2 resources have been changed since the last commit point. In a CICS or RRSAF application, the two-phase commit process begins only if DB2 resources have changed and a non-DB2 resource has changed within the same commit scope. For more information on the commit process for IMS and CICS applications, see “Consistency with other systems” on page 359.

The significant events that show up in a performance trace of a commit and thread termination operation are as follows:

1. Commit phase 1

   In commit phase 1 (IFCID 0084), DB2 writes an end of phase 1 record to the log (IFCIDs 0032 and 0033). There are two I/Os, one to each active log data set (IFCIDs 0038 and 0039).

2. Commit phase 2

   In commit phase 2 (IFCID 0070), DB2 writes a beginning of phase 2 record to the log. Again, the trace shows two I/Os. Page and row locks (except those protecting the current position of cursors declared with the WITH HOLD option), held to a commit point, are released. An unlock (IFCID 0021) with a requested token of zeros frees any lock for the specified duration. A summary lock record (IFCID 0020) is produced, which gives the maximum number of page locks held and the number of lock escalations. DB2 writes an end of phase 2 record to the log.

   If RELEASE(COMMIT) is used, the following events also occur:

   - Table space locks are released.
   - All the storage used by the thread is freed, including storage for control blocks, CTs and PTs, and working areas.
   - The use counts of the DBDs are decreased by one. If space is needed in the EDM pool, a DBD can be freed when its use count reaches zero.
   - Those table spaces and index spaces with no claimers are made candidates for deferred close. See “Understanding the CLOSE YES and CLOSE NO options” on page 595 for more information on deferred close.

3. Thread termination

   When the thread is terminated, the accounting record is written. It does not report transaction activity that takes place before the thread is created.

   If RELEASE(DEALLOCATE) is used to release table space locks, the DBD use count is decreased, and the thread storage is released.

## Variations on thread management

Minor differences exist in the transaction flow in different environments and for SQL statements originating dynamically.

### TSO and call attachment facility differences

The TSO attachment facility and call attachment facility (CAF) can be used to request that SQL statements from CICS, IMS, or RRSAF be executed in TSO foreground and batch. The processes differ from CICS or IMS transactions in that:

- There is no sign-on. The user is identified when the TSO address space is connected.
- Commit requires only a single phase and only one I/O operation to each log. Single phase commit records are IFCID 0088 and 0089.
- Threads cannot be reused, because the thread is allocated to the user address space.

### Thread management for Recoverable Resource Manager Services Attachment Facility (RRSAF)

With RRSAF, you have sign-on capabilities, the ability to reuse threads, and the ability to coordinate commit processing across different resource managers. For more information, see Part 6 of *DB2 Application Programming and SQL Guide*.

### Differences for SQL under QMF

QMF uses CAF to create a thread when a request for work, such as a SELECT statement, is issued. A thread is maintained until the end of the session only if the requester and the server reside in different DB2 subsystems. If the requester and the server are both in the local DB2 subsystem, the thread is not maintained.

For more information on QMF connections, see *Query Management Facility: Installing and Managing QMF on OS/390 and z/OS*.

## Providing for thread reuse

In general, you want transactions to reuse threads when transaction volume is high and the cost of creating threads is significant, but thread reuse is also useful for a lower volume of priority transactions. For a transaction of five to ten SQL statements (10 I/O operations), the cost of thread creation can be 10% of the processor cost. But the steps needed to reuse threads can incur costs of their own.

Later in this chapter, the following sections cover thread reuse for specific situations:

- “Reusing threads for remote connections” on page 629 provides information on the conditions for thread reuse for database access threads.
- “CICS design options” on page 633 tells how to write CICS transactions to reuse threads.
- “IMS design options” on page 639 tells how to write IMS transactions to reuse threads.

### Bind options for thread reuse

In DB2, you can prepare allied threads for reuse by binding the plan with the ACQUIRE(USE) and RELEASE(DEALLOCATE) options; otherwise, the allocation cost is not eliminated but only slightly reduced. Be aware of the following effects:

- ACQUIRE(ALLOCATE) acquires all resources needed by the plan, including locks, when the thread is created; ACQUIRE(USE) acquires resources only when they are needed to execute a particular SQL statement. If most of the SQL statements in the plan are executed whenever the plan is executed, ACQUIRE(ALLOCATE) costs less. If only a few of the SQL statements are likely to be executed, ACQUIRE(USE) costs less and improves concurrency. But with thread reuse, if most of your SQL statements eventually get issued, ACQUIRE(USE) might not be as much of an improvement.

- RELEASE(DEALLOCATE) does not free cursor tables (SKCTs) at a commit point; hence, the cursor table could grow as large as the plan. If you are using created temporary tables, the logical work file space is not released until the thread is deallocated. Thus, many uses of the same created temporary table do not cause reallocation of the logical work files, but be careful about holding onto this resource for long periods of time if you do not plan to use it.

### Using reports to tell when threads were reused

The NORMAL TERM., ABNORMAL TERM., and IN DOUBT sections of the DB2 PM accounting report, shown in Figure 70, can help you identify, by plan, when threads were reused. In the figure:

- NEW USER ( A ) tells how many threads were not terminated at the end of the previous transaction or query, and hence reused.
- DEALLOCATION ( B ) tells how many threads were terminated at the end of the query or transaction.
- APPL. PROGR. END ( C ) groups all the other reasons for accounting. Since the agent did not abend, these are considered normal terminations.

This technique is accurate in IMS but not in CICS, where threads are reused frequently by the same user. For CICS, also consider looking at the number of commits and aborts per thread. For CICS:

- NEW USER ( A ) is thread reuse with a different authorization ID or transaction code.
- RESIGN-ON ( D ) is thread reuse with the same authorization ID if TOKENE=YES.

```
NORMAL TERM.      TOTAL  ABNORMAL TERM.      TOTAL  IN DOUBT            TOTAL
----------------  -----  ------------------  -----  -----------------   ------
NEW USER A           17  APPL.PROGR. ABEND       0  APPL.PGM. ABEND          0
DEALLOCATION B        0  END OF MEMORY           0  END OF MEMORY            0
APPL.PROGR. END C     0  RESOL.IN DOUBT          0  END OF TASK              0
RESIGNON  D           0  CANCEL FORCE            0  CANCEL FORCE             0
DBAT INACTIVE         0
RRS COMMIT            0
```

*Figure 70. DB2 PM accounting report - information about thread termination*

## Database access threads

This section describes:

- “Understanding allied threads and database access threads” on page 625
- “Setting thread limits for database access threads” on page 625
- “Using inactive threads” on page 626
- “Establishing a remote connection” on page 628
- “Reusing threads for remote connections” on page 629
- “Using Workload Manager to set performance objectives” on page 629

For information on performance considerations for distributed processing, see “Tuning distributed applications” on page 858.

## Understanding allied threads and database access threads

Database access threads are created to access data at a DB2 server on behalf of a requester using either DRDA or DB2 private protocol. A database access thread is created when an SQL request is received from the requester. Allied threads perform work at a requesting DB2.

Database access threads differ from allied threads in the following ways:

- Database access threads can be always active or both active and inactive, depending on what you specified for the DDF THREADS field on installation panel DSNTIPR.
- Database access threads run in enclave SRB mode.
- When database access threads can be both active and inactive, a thread is terminated after 200 transactions have used it, or after the thread has been idle in the pool for the amount of time specified in the POOL THREAD TIMEOUT field on installation panel DSNTIP5.
- If inbound translation is used, the sign-on audit trace record (0087) is cut to audit the change of authorization IDs. There is no BEGIN SIGN-ON (0086) record in this case.

## Setting thread limits for database access threads

When you install DB2, you choose a maximum number of active threads that can be allocated concurrently; the MAX USERS field on panel DSNTIPE represents the maximum number of allied threads, and the MAX REMOTE ACTIVE field on panel DSNTIPE represents the maximum number of database access threads. Together, the values you specify for these fields cannot exceed 2000.

In the MAX REMOTE CONNECTED field of panel DSNTIPE, you can specify up to 150 000 as the maximum number concurrent remote connections that can concurrently exist within DB2. This upper limit is only obtained if you specify the recommended value INACTIVE for the DDF THREADS field of installation panel DSNTIPR. Figure 71 on page 626 illustrates the relationship among the number of active threads in the system and the total number of connections.

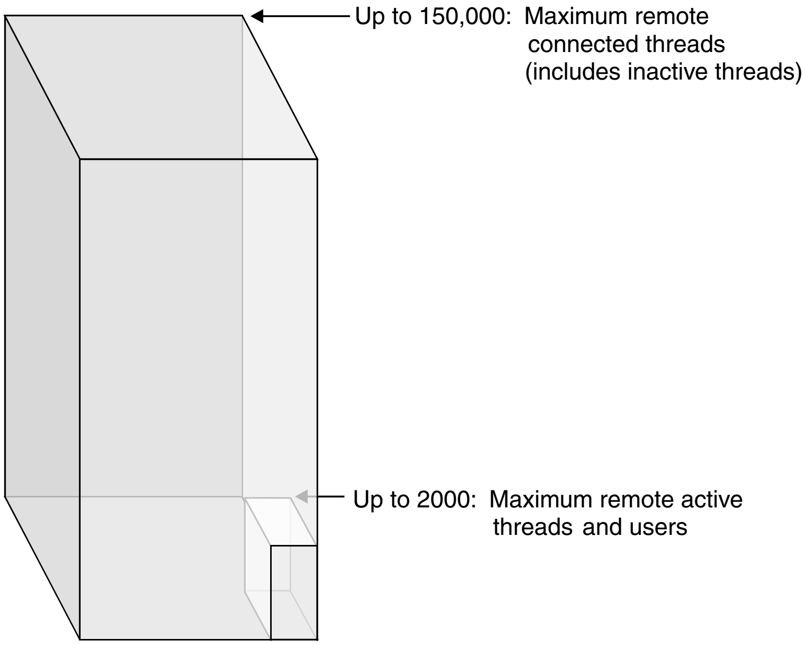

Up to 150,000: Maximum remote connected threads (includes inactive threads)
Up to 2000: Maximum remote active threads and users

*Figure 71. Relationship between active threads and maximum number of connections.*

## Using inactive threads

A database access thread that does not hold any cursors is called an *inactive thread*. (Because inactive threads can become active, another name for these types of threads might be *sometimes active*.) DB2 supports two types of inactive threads: *type 1* and *type 2*. The differences between the types are that type 2 inactive threads are only available for DRDA connections, use less storage than type 1 inactive threads, and use a pool of database access threads that can be switched among connections as needed.

### Using type 2 inactive threads

DB2 always tries to make inactive threads type 2, but in some cases cannot do so. The conditions listed in Table 88 determine if a thread can be a type 2 or a type 1.

*Table 88. Requirements for type 1 and type 2 inactive threads*

| If there is... | Thread can be type 2? | Thread can be type 1? |
|---|---|---|
| A hop to another location | Yes | Yes |
| A connection using DB2 private—protocol access | No | Yes |
| A package that is bound with RELEASE(COMMIT) | Yes | Yes |
| A package that is bound with RELEASE(DEALLOCATE) | Yes | No |
| A held cursor, a held LOB locator, or a package bound with KEEPDYNAMIC(YES) | No | No |
| A declared temporary table that is active (the table was not explicitly dropped through the DROP TABLE statement) | No | No |

When the conditions listed in Table 88 on page 626 are true, the thread can become inactive when a COMMIT is issued. After a ROLLBACK, a thread can become inactive even if it had open cursors defined WITH HOLD or a held LOB locator because ROLLBACK closes all cursors and LOB locators.

## Determining if a thread can become inactive

After a COMMIT or ROLLBACK, DB2 determines if a thread can become inactive and, if so, if that thread can become a type 1 or type 2 inactive thread based on the conditions shown in Table 88 on page 626.

If a thread is eligible to become a type 2 inactive thread, the thread is made inactive and the database access thread is eligible to be used by another connection.

If a thread must become a type 1 inactive thread, DB2 first compares the number of current type 1 inactive threads to the value that is specified for your installation for MAX TYPE 1 INACTIVE and either makes the thread inactive or allows it to remain active:

1. If the current number of type 1 inactive threads is below the value in MAX TYPE 1 INACTIVE, the thread becomes inactive. It cannot be used by another connection.
2. If the current number of type 1 inactive threads meets or exceeds the value in MAX TYPE 1 INACTIVE, the thread remains active. However, too many active threads (that is, more than MAX REMOTE ACTIVE) can cause the thread and its connection to be terminated.

## Understanding the advantages of inactive threads

Letting threads go inactive has the following advantages:

- You can leave an application that is running on a workstation connected to DB2 from the time the application is first activated until the workstation is shut down and thus avoid the delay of repeated connections.
- DB2 can support a larger number of DDF threads (150 000 instead of 1999).
- Less storage is used for each DDF thread. (A type 2 inactive thread uses significantly less storage than a type 1 inactive thread.)
- You get an accounting trace record (IFCID 0003) each time a thread becomes inactive rather than once for the entire time you are connected. When an inactive thread becomes active, the accounting fields for that thread are initialized again. As a result, the accounting record contains information about active threads only. This makes it easier to study how distributed applications are performing.
- Each time a thread becomes inactive, workload manager resets the information it maintains on that thread. The next time that thread is activated, workload manager begins managing to the goals you have set for transactions that run in that service class. If you use multiple performance periods, it is possible to favor short-running units of work that use fewer resources while giving fewer resources over time to long running units of work. See “Establishing performance periods for DDF threads” on page 632 for more information.
- If using WLM goal mode, you can use response time goals, which is not recommended when using threads that are always active.
- It makes it more practical to take advantage of the ability to time out idle active threads, as described in “Timing out idle active threads” on page 628.
- Type 2 inactive threads can be reused, as they can be in CICS and IMS. Thread reuse lets DDF use a small pool of DB2 threads to support a large group of network clients.

- The response times reported by RMF include inactive periods between requests. These times are shown as idle.

### Enabling threads to become inactive

You must specify INACTIVE on the DDF THREADS field of installation panel DSNTIPR to allow threads to become inactive. To limit the number of type 1 inactive threads that can be created, specify a value in the MAX TYPE 1 INACTIVE field of installation panel DSNTIPR. The default is 0, which means that any thread that does not qualify for being a type 2 inactive thread remains active.

**Recommendation:** Use type 2 inactive threads if you can. If you can't, set MAX TYPE 1 INACTIVE to the maximum number of concurrent connections that access another location using private—protocol access.

### Timing out idle active threads

Active server threads that have remained idle for a specified period of time (in seconds) can be canceled by DB2. When you install DB2, you choose a maximum IDLE THREAD TIMEOUT period, from 0 to 9999 seconds. The timeout period is an approximation. If a server thread has been waiting for a request from the requesting site for this period of time, it is canceled unless it is an inactive or an in doubt thread. A value of 0, the default, means that the server threads cannot be canceled because of an idle thread timeout.

**Recommendation:** Use this option with the option INACTIVE for the DDF THREADS field on DSNTIPR. If you specify a timeout interval with ACTIVE, an application would have to start its next unit of work within the timeout period specification, or risk being canceled.

***TCP/IP keep_alive interval for the DB2 subsystem:***For TCP/IP connections, it is a good idea to specify the IDLE THREAD TIMEOUT value in conjunction with a TCP/IP keep_alive interval of 5 minutes or less to make sure that resources are not locked for a long time when a network outage occurs. You can override the TCP/IP stack keep_alive interval on a single DB2 subsystem by specifying a value in the field TCP/IP KEEPALIVE on installation panel DSNTIPS.

## Establishing a remote connection

The following steps occur when establishing a connection for a database access thread:

1. Connection or signon.

   SNA network connections support connection and signon processing. TCP/IP network connections support just connection processing.
2. If you specified INACTIVE for the DDF THREADS, DB2 checks the MAX REMOTE CONNECTED limit you specified on panel DSNTIPE to see if it has been reached.

   If the limit has been reached, DB2 does not create an active or an inactive thread; the create thread request is rejected, and the connection is deallocated.

   If the MAX REMOTE CONNECTED limit has not been reached, the connection process continues.
3. DB2 compares the MAX REMOTE ACTIVE limit you specified on panel DSNTIPE with the current number of active database access threads.
   - If the MAX REMOTE ACTIVE limit is reached, DB2 queues the connection request until an unused database access thread can be assigned to establish the remote connection. When an unused database access thread becomes available, the connection is established.

- If the MAX REMOTE ACTIVE limit is not reached, the connection is established and a database access thread is created.

4. DB2 verifies the user through DCE, RACF or the communications database.

   SNA network connections support DCE, RACF, or the communications database. TCP/IP network connections support DCE or RACF user verification.
5. DB2 checks the user’s authorization to connect to DDF through RACF or the communications database.

   SNA network connections can use RACF or the communications database to check authorization. TCP/IP network connections can use RACF to check authorization.
6. If the connection is using SNA, DB2 can use the communications database to translate the remote user ID to a DB2 authorization ID.
7. DB2 creates the MVS enclave.

The “Global DDF Activity” section of the DB2 PM statistics report shows information about database access threads.

## Reusing threads for remote connections

The cost to create a thread can be significant and, as described in “Providing for thread reuse” on page 623, reusing threads is a way to avoid that cost. DB2 for OS/390 and z/OS can reuse threads at the requester and at the server. At the requester, a thread can be reused for an application that uses the CICS, IMS, or RRS attachment facility as described later in this chapter. As a server, DB2 can assign that connection to a thread from among a pool of type 2 inactive threads. Those threads can be shared and reused by thousands of client connections, which lets DB2 support very large client networks at minimal cost. (Type 1 inactive threads are only eligible to be reused by the same connection.)

If your server is not DB2 for OS/390 and z/OS, or some other server that can reuse threads, then reusing threads for your requesting CICS, IMS, or RRS applications is not a benefit for distributed access. Thread reuse occurs when sign-on occurs with a new authorization ID. If that request is bound for a server that does not support thread reuse, that change in the sign-on ID causes the connection between the requester and server to be released so that it can be rebuilt again for the new ID.

## Using Workload Manager to set performance objectives

MVS supports enclave system request blocks (SRBs). An MVS enclave lets each thread have its own performance objective. Using MVS’s workload management support, you can establish MVS performance objectives for individual DDF server threads, including threads that run in WLM-established stored procedures address spaces. (Stored procedures that run in the DB2-established stored procedures address space always run at the performance objective of that address space.) For details on using workload management, see *OS/390 MVS Planning: Workload Management*.

The MVS performance objective of the DDF address space or the WLM-established stored procedures address spaces does not govern the performance objective of the user thread. As described in “MVS performance options for DB2” on page 614, you should assign the DDF address space and WLM-established stored procedures address spaces to an MVS performance objective that is similar to the DB2 database services address space (*ssnm*DBM1). The MVS performance objective of the DDF and stored procedures address spaces determines how quickly DB2 is able to perform operations associated with managing the distributed DB2 work load, such as adding new users or removing users that have terminated their connections.

Workload manager has two modes:

- Compatibility mode
- Goal mode

Many of the concepts and actions required to manage enclaves are common to both compatibility and goal modes; those are described first. Considerations specific for compatibility mode are described in “Considerations for compatibility mode” on page 632.

**Attention:** If you do not classify your DDF transactions into service classes, they are assigned to the default class, the *discretionary* class, which is at a very low priority.

## Classifying DDF threads

You can classify DDF threads by, among other things, authorization ID and stored procedure name. The stored procedure name is only used as a classification if the first statement issued by the client after the CONNECT is an SQL CALL statement. Use the workload manager administrative application to define the service classes you want MVS to manage. These service classes are associated with performance objectives. When a WLM-established stored procedure call originates locally, it inherits the performance objective of the caller, such as TSO or CICS.

***Classification attributes:*** Each of the WLM classification attributes has a two or three character abbreviation that you can use when entering the attribute on the WLM menus. The following WLM classification attributes pertain to DB2 DDF threads:

**AI** Accounting information. The value of the DB2 accounting string associated with the DDF server thread, described by QMDAAINF in the DSNDQMDA mapping macro.

**CI** The DB2 correlation ID of the DDF server thread, described by QWHCCV in the DSNDQWHC mapping macro.

**CN** The DB2 collection name of the *first* SQL package accessed by the DRDA requester in the unit of work.

**LU** The VTAM LUNAME of the system that issued the SQL request.

**NET** The VTAM NETID of the system that issued the SQL request.

**PC** Process name. This attribute can be used to classify the application name or the transaction name. The value is defined by QWHCEUTX in the DSNDQWHC mapping macro.

**PK** The name of the *first* DB2 package accessed by the DRDA requester in the unit of work.

**PN** The DB2 plan name associated with the DDF server thread. For DB2 private protocol requesters and DB2 DRDA requesters that are at Version 3 or subsequent releases, this is the DB2 plan name of the requesting application. For other DRDA requesters, use ’DISTSERV’ for PN.

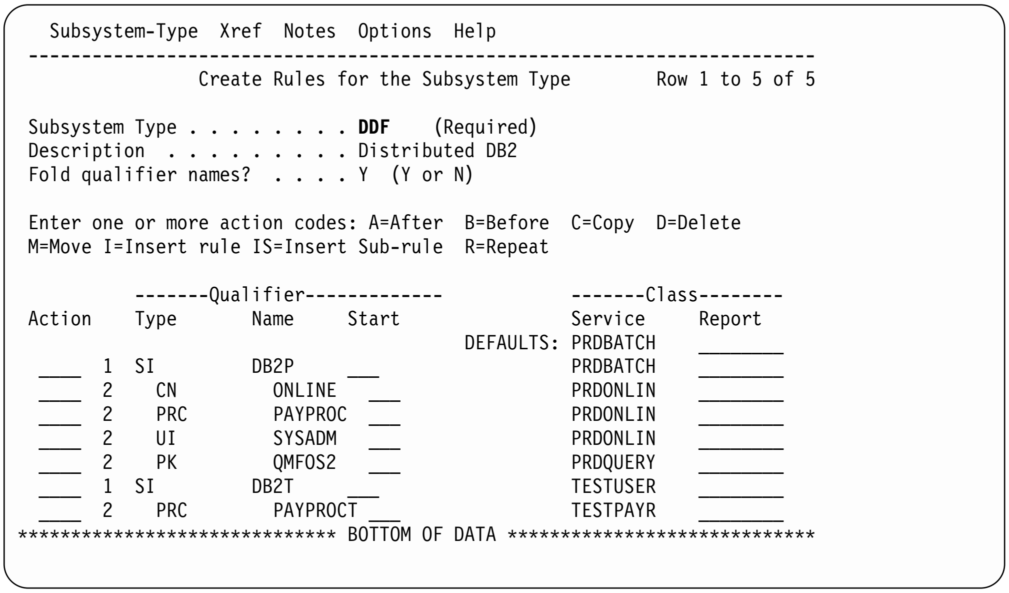
**PRC** Stored procedure name. This classification only applies if the first SQL statement from the client is a CALL statement.

**SI** Subsystem instance. The DB2 server’s MVS subsystem name.

**SSC** Subsystem collection name. When the DB2 subsystem is a member of a DB2 data sharing group, this attribute can be used to classify the data sharing group name. The value is defined by QWHADSGN in the DSNDQWHA mapping macro.

**UI** User ID. The DDF server thread’s primary authorization ID, after inbound name translation.

Figure 72 shows how you can associate DDF threads and stored procedures with service classes.



```
   Subsystem-Type  Xref  Notes  Options  Help
 --------------------------------------------------------------------------
                 Create Rules for the Subsystem Type        Row 1 to 5 of 5

 Subsystem Type . . . . . . . . DDF    (Required)
 Description  . . . . . . . . . Distributed DB2
 Fold qualifier names?  . . . . Y  (Y or N)

 Enter one or more action codes: A=After  B=Before  C=Copy  D=Delete
 M=Move I=Insert rule IS=Insert Sub-rule  R=Repeat

           -------Qualifier-------------            -------Class--------
 Action    Type       Name     Start                Service     Report
                                          DEFAULTS: PRDBATCH    ________
  ____  1  SI         DB2P     ___                  PRDBATCH    ________
  ____  2    CN         ONLINE   ___                PRDONLIN    ________
  ____  2    PRC        PAYPROC  ___                PRDONLIN    ________
  ____  2    UI         SYSADM   ___                PRDONLIN    ________
  ____  2    PK         QMFOS2   ___                PRDQUERY    ________
  ____  1  SI         DB2T     ___                  TESTUSER    ________
  ____  2    PRC        PAYPROCT ___                TESTPAYR    ________
****************************** BOTTOM OF DATA ******************************
```

*Figure 72. Classifying DDF threads using Workload Manager. You assign performance goals to service classes using the services classes menu of WLM.*

In Figure 72, the following classifications are shown:

- All DB2P applications accessing their first SQL package in the collection ONLINE are in service class PRDONLIN.
- All DB2P applications that call stored procedure PAYPROC first are in service class PRDONLIN.
- All work performed by DB2P user SYSADM is in service class PRDONLIN.
- Users other than SYSADM that run the DB2P PACKAGE QMFOS2 are in the PRDQUERY class. (The QMFOS2 package is not in collection ONLINE.
- All other work on the production system is in service class PRBBATCH.
- All users of the test DB2 system are assigned to the TESTUSER class except for work that first calls stored procedure PAYPROCT, which is in service class TESTPAYR.

***Don’t create too many stored procedures address spaces:*** Workload manager creates one or more stored procedures address spaces for every combination of caller’s service class and WLM environment name for which work exists. The number of tasks in an address space is also specified to help control the number of address spaces created. See “Assigning procedures and functions to WLM application environments” on page 875 for more information.

## Establishing performance periods for DDF threads

You can establish performance periods for DDF threads, including threads that run in the WLM-established stored procedures address space. By establishing multiple performance periods, you can cause the thread's performance objectives to change based upon the thread's processor consumption. Thus, a long-running unit of work can move down the priority order and let short-running transactions get in and out at a higher priority.

To design performance strategies for these threads, take into account the events that cause a DDF thread to reset its MVS performance period. The MVS performance period is reset by terminating the MVS enclave for the thread and creating a new MVS enclave for the thread, as described in "Using RMF to monitor distributed processing" on page 870.

Because threads that are always active do not terminate the enclave and thus do not reset the performance period to period 1, a long-running thread will always end up in the last performance period. Any new business units of work that use that thread will suffer the performance consequences. This makes performance periods unattractive for long-running threads. For always active threads, therefore, use velocity goals and use a single-period service class.

## Basic procedure for establishing performance objectives

To establish performance objectives for DDF threads and the related address spaces, use the following steps:

1. Create a workload manager service definition that assigns service classes to the DDF threads under subsystem type DDF and to the DDF address space under subsystem type STC. If you are using WLM-established stored procedures address spaces, assign a service class to them under subsystem type STC.
2. Install the service definition using the MVS workload manager menus and activate a policy (VARY WLM,POLICY=*policy*).
3. If your system is running in compatibility mode, follow the additional steps described in "Considerations for compatibility mode".

## Considerations for compatibility mode

In compatibility mode, threads are given a service class by the classification rules in the active WLM service policy. The MVS ICS maps service classes (SRVCLASS) to a performance group number (PGN), which determines the performance group of the enclave. When workload manager operates in compatibility mode, take the following actions to establish performance objectives for DDF threads:

1. Define MVS performance groups (PGNs) for DDF threads in the IPS PARMLIB member. Do the same for WLM-established stored procedures address spaces if you are using them.
2. Create MVS ICS PARMLIB definitions to map the service classes assigned in the workload manager classification rules to the corresponding performance groups, using SUBSYS=DDF and the SRVCLASS keyword. The subsystem default performance group for SUBSYS=DDF is ignored.
3. Create MVS PARMLIB definitions to assign a performance group to the WLM-established stored procedures address spaces if you are using them. The same performance group can be assigned to these stored procedures address spaces as is assigned to DDF.
4. Activate the updated parmlib members (SET IPS=*xx*, ICS=*yy*).

Each of the PGN values in the MVS ICS must be defined in the IPS PARMLIB member. The PGN definition can include information on the performance period, which is used by SRM to change the performance objective of a DDF thread based on the amount of processor resource the DDF thread consumes.

***Stored procedures and user-defined functions:*** When you run in compatibility mode, you have to take on more performance management issues. With functions and procedures that run in WLM-established address spaces, for example, WLM cannot automatically start a new address space to handle additional high-priority requests, as it can when using goal mode. You must monitor the performance of the stored procedures and user-defined functions to determine how many WLM-managed address spaces to start manually.

### Considerations for goal mode

In goal mode, threads are assigned a service class by the classification rules in the active WLM service policy. Each service class period has a performance objective (goal), and workload manager raises or lowers that period's access to system resources as needed to meet the specified goal. For example, the goal might be "application APPL8 should run in less than 3 seconds of elapsed time 90% of the time".

Assign the DDF address space and any WLM-established address spaces for stored procedures and user-defined functions to the same service class as the DB2 database services address space (*ssnm*DBM1). Define this service class with a velocity goal.

***Stored procedures and user-defined functions:*** When you are in goal mode, WLM automatically starts WLM-established address spaces for stored procedures and user-defined functions to help meet the service class goals you set. This is assuming you have defined the application environment, as described in "Assigning procedures and functions to WLM application environments" on page 875.

No matter what your service class goals are, it is possible for the request to start an address space to time out, based on the timeout value you specify on the TIMEOUT VALUE field of installation DSNTIPX. If the timeout value is too small, you might need to increase it to account for a busy system.

## CICS design options

This section applies to CICS Transaction Server for OS/390 Release 1 and previous versions of CICS. If you are using CICS Transaction Server for OS/390 Release 2, or later releases, see:

- *CICS Resource Definition Guide* for information on RDO (Resource Definition Online) definition of the resource control table (RCT)
- *CICS DB2 Guide* for information about DB2 performance considerations and setup of the CICS attachment facility

The information under this heading, up to "IMS design options" on page 639, is Product-sensitive Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1095.

This section includes the following topics:

- "Overview of RCT options" on page 634
- "Plans for CICS applications" on page 634
- "Thread creation, reuse, and termination" on page 634
- "Recommendations for RCT definitions" on page 637
- "Recommendations for accounting information for CICS threads" on page 639

## Overview of RCT options

You can tune your CICS attachment facility by entering values in the resource control table (RCT) with the following macros and options:

- DSNCRCT TYPE=INIT macro:

  **THRDMAX** The maximum total number of CICS DB2 threads.

  **PURGEC** The normal length of the purge cycle, specified in minutes and seconds.

  **TXIDSO** User sign-on preferences with transaction ID changes.

  **TOKENI** See the description of TOKENE, below.

- DSNCRCT TYPE=ENTRY and TYPE=POOL macros:

  **DPMODE** Thread TCB priority relative to the CICS main TCB.

  **THRDM** The maximum number of threads.

  **THRDA** The current maximum number of threads. This value can be changed dynamically, up to the value specified in THRDM.

  **THRDS** The number of protected threads.

  **TWAIT** The transaction disposition when THRDA has already been reached (wait, abend, or divert to the pool).

  **AUTH** The authorization ID to be used by the CICS attachment facility when signing on to DB2.

  **TOKENE=(YES|NO)**
  YES means that DB2 produces an accounting record for every CICS transaction, even those transactions that are reusing threads. For more information about using TOKENE, see "Recommendations for accounting information for CICS threads" on page 639.

For more information about specifying the CICS attachment facility macros, see Part 2 of *DB2 Installation Guide*.

## Plans for CICS applications

You can use either packages or dynamic plan selection to manage your CICS applications, but packages offer more flexibility. See *DB2 Application Programming and SQL Guide* for more information about using packages. See "Routines for dynamic plan selection in CICS" on page 946 for more information about writing dynamic plan selection exit routines.

## Thread creation, reuse, and termination

A thread is a structure that allows a non-DB2 address space to request work from DB2. CICS threads are anchored in a TCB. The CICS attachment facility sets up a number of TCBs in the CICS address space that application programs can use for SQL calls.

***Types of threads:*** The attachment facility has 2 types of threads:

- An *unprotected thread* is terminated immediately after the transaction is through with it (at SYNCPOINT or EOT). An unprotected thread can be reused before it is terminated if a waiting transaction (TWAIT=YES) uses the same plan.

- A *protected thread* remains for a time after the transaction is through with it to increase the chances of thread reuse. That time is determined by the purge cycle, normally 30 seconds.

***States of threads:*** The following terms identify the state a thread is in:

- *Identified* indicates that the TCB is known to DB2.
- *Signed on* indicates that DB2 has processed and approved the authorization ID for the thread for the plan name.
- *Created* indicates that DB2 has allocated the plan and can process the SQL requests.

You can see these various states when you issue the DB2 command DISPLAY THREAD. See Figure 25 on page 293 for an example of how CICS threads appear in the output.

It is possible for a thread that has been created to be signed on again without re-creating the thread. This is known as reusing the thread.

***Number of threads:*** To limit the number of threads in a CICS environment, you should limit the transactions from CICS before they make DB2 requests. Controls in CICS determine how many tasks can be created for a transaction class. Use these controls to limit the number of CICS tasks accessing DB2 to the number of available threads as determined by the value in the MAX USERS field of installation panel DSNTIPE. By limiting this number, you avoid having threads queue at create thread time. See “Recommendations for CICS system definitions” on page 639 for information.

## When CICS threads are created

When a transaction needs a thread, an existing thread can be reused, or a new thread can be created. If no existing thread is available, and if the maximum number of threads (THRDA) has not been reached, a thread is created.

This section describes both the creation of the TCBs and the sign-on activity.

***Creating thread TCBs:*** When the CICS attachment facility is started, some TCBs could be attached to threads for each RCT entry. The number of TCBs for each entry attached when the CICS attachment facility is started is given by THRDS. Those threads are protected.

If THRDA is greater than THRDS, some TCBs are not attached when the attachment facility is started, but only when needed by a task. The number of TCBs for each entry attached only when needed by a thread is given by THRDA - THRDS. Those threads are not protected.

***Sign-on processing:*** Threads sign on for the following reasons:

- To tell DB2 who the user is
- To create accounting trace records
- To put a thread back into its initial state.

Sign-on occurs at the first SQL call when any of the following is true:

- The authorization ID changed.
- The transaction ID changed and TXIDSO=YES.
- The parameter TOKENE is YES. For more information about TOKENE, see “Recommendations for accounting information for CICS threads” on page 639.

- The last transaction left a held cursor open.
- The last transaction left one of the modifiable special registers in use.
- The last transaction is holding a LOB locator.

***Using TXIDSO to control sign-on processing:*** With CICS, you can use the option TXIDSO in the RCT with TYPE=INIT to specify your preference for sign-on:

- TXIDSO=YES means that the thread must sign-on even when the only thing that has changed is the transaction ID.
- TXIDSO=NO means that if only the transaction ID has changed, the thread can be reused with no sign-on.

This option affects only pool threads and those RCT entry threads with multiple transaction IDs in one entry.

## When CICS threads are released and available for reuse

An existing thread can be reused by a new transaction with the same plan and on the same RCT entry. The thread is released for reuse (or for termination) at the end of a task (EOT) or at SYNCPOINT.

- A transaction that is not terminal-driven releases its thread at the end of a task.
- A transaction that is terminal-driven can release its thread at SYNCPOINT, if certain conditions are true. DB2 uses the following logic to determine whether a thread can be released at SYNCPOINT, or if it must wait until EOT:
  1. Is the thread terminal-driven?

     If the answer is yes, go to the next step. If the answer is no, the thread cannot be released until EOT.
  2. Are the following special registers in their initial state?

     CURRENT APPLICATION ENCODING SCHEME
     CURRENT PACKAGESET
     CURRENT PRECISION
     CURRENT RULES
     CURRENT SERVER
     CURRENT SQLID

     If the answer is yes, go to the next step. If the answer is no, the thread cannot be released until EOT.
  3. Has the special register CURRENT DEGREE never been changed during the life of the thread?

     If it has not been changed, go to the next step. If it has been changed, the thread cannot be released until EOT.
  4. Have all declared temporary tables defined as ON COMMIT PRESERVE ROWS been explicitly dropped by using the DROP TABLE statement?

     If the answer is yes, go to the next step. If the answer is no, the thread cannot be released until EOT.
  5. Are all cursors declared WITH HOLD closed?

     If the answer is yes, this thread can be released at SYNCPOINT. If the answer is no, and this is a local connection, this thread cannot be released until EOT.

     If the answer is no, and this is a remote connection, look at the DISCONNECT bind option:

     **AUTOMATIC**
     All connections (even those with open cursors) are released at commit, and the thread can be released. However, the thread is NOT

reusable if you have a type 1 connection and the value of the BIND option CURRENTSERVER is a remote location.

**EXPLICIT**
Does the application use the SQL statement RELEASE ALL? If the answer is yes, the thread can be released. If the answer is no, the thread cannot be released until EOT.

**CONDITIONAL**
The thread cannot be reused until EOT if there are any open cursors defined WITH HOLD.

### When CICS threads terminate

This section describes when the two types of threads terminate.

***Protected thread termination:*** When a protected thread (TYPE=ENTRY) is released, it waits for two consecutive purge cycles and terminates if it is unused at the end of the second purge cycle.

The purge cycle is 5 minutes long when the CICS attachment facility first initializes. With CICS Version 4 (or later), you can determine the length of the normal purge cycle using the RCT parameter PURGEC=(minutes,seconds). The maximum specifiable length of a purge cycle is 59 minutes, 59 seconds. The minimum length is 30 seconds, which is the default.

Threads remain available for reuse for an average of (purge cycle time × 1.5).

***Unprotected thread termination:*** Unprotected threads terminate as soon as the thread is released, unless another transaction with the same plan is queued for the thread.

***TCB detachment:*** After a TCB has been attached to a thread, the TCB is available until the attachment facility is stopped. TCBs are detached only when the number of active TCBs reaches THRDMAX - 2. Thus when the thread is terminated, the associated TCB is not detached.

## Recommendations for RCT definitions

Set the following RCT parameters:

- Make the sum of the THRDA values from all COMD, ENTRY, and POOL threads less than THRDMAX - 2. Otherwise, a thread and its associated TCB, whether protected or not, are terminated when the number of threads is THRDMAX - 2. If not explicitly specified, the COMD thread has a default THRDA value of one.
- For TYPE=POOL, set THRDA equal to the sum of the expected number of threads for the pool. THRDA should be the sum of:
  - Transactions with THRDA=0 that are forced to the pool
  - Transactions that can overflow to the pool
  - Transactions defined by the pool
- Use TYPE=ENTRY with THRDA=*n* and THRDS=*n* for high volume transaction groups. Those transactions reuse threads. If queuing for a thread is acceptable, use TWAIT=YES and make *n* large enough to handle the normal transaction load with minimal queuing. If queuing for a thread is not acceptable, use TWAIT=POOL.
- Use TYPE=ENTRY with THRDA=*n*, THRDS=0, and TWAIT=YES for a transaction or group for which you want to do any of the following:

- Control the maximum number of concurrent transactions, *n*. If *n* is 1, you are serializing the transaction or group. You can achieve similar results with the CICS controls, as described in “Recommendations for CICS system definitions” on page 639.
  - Force serialization.
  - Avoid “flooding” the pool threads with possibly high-volume transactions.
  - Provide dedicated entries for high priority transactions with a volume that does not justify the use of protected threads. However, compared to a THRDS>0 entry, you are not likely to achieve thread reuse unless the transaction rate is high. In this case, using some number of protected entry threads might be a better choice.
- For transactions that can use default TYPE=POOL parameters, allow them to default to the pool. The fewer TYPE=ENTRY definitions you have, the less maintenance there is on the RCT.
- Use TYPE=ENTRY with THRDA=0, THRDS=0, and TWAIT=POOL for those transactions that need something special besides the default TYPE=POOL definitions. For example, you might want a transaction to run in the POOL but use TOKENE=YES.

***Setting thread TCB priority using DPMODE:*** The RCT DPMODE parameter controls the priority of the thread TCBs. In general, specify the default DPMODE=HIGH for high-priority and high-volume transactions. The purpose is to execute these transactions quickly, removing them from CICS and DB2. This helps save virtual storage, and allows the transaction to release its locks to avoid causing other transactions to deadlock or timeout.

However, if there is a risk that one or more SQL statements in the transaction will consume a great deal of processor time, allowing the thread TCB to monopolize the processor, the CICS main TCB might not be dispatched. (Processor monopolization such as this causes the most impact on single-CP machines.)

The result of concurrent high priority CICS activity in DB2 can cause transactions to appear to run longer in DB2. In such cases, CICS tracing shows the task as “waiting for a DB2 ECB”, while the DB2 accounting trace reports the task as “not in DB2” time. The reason this occurs is that CICS has not had a chance to dispatch the task that DB2 has posted or the task is waiting for a thread to become available.

Do not misread this situation and then set DPMODE=HIGH, because then the problem will get worse. Instead, weigh the importance of the concurrent CICS activity versus the DB2 activity and adjust the task priorities and the DPMODE setting accordingly (DPMODE=LOW or DPMODE=EQUAL).

***Recommendations for DPMODE:*** In general, use the following:

- DPMODE=HIGH for high-priority and high-volume transactions
- DPMODE=EQUAL for transactions that are more CICS-intensive than DB2-intensive (such as short, simple SQL statements)
- DPMODE=LOW for long running and low-priority, short SQL transactions, especially non-terminal-driven transactions.

## Recommendations for CICS system definitions

The following specification controls how many tasks can be created for a transaction class: CEDA DEFINE TRANCLASS() GROUP() MAXACTIVE() in the CSD.

Use the following recommendations for setting CMXT or MAXACTIVE:

- When TWAIT=YES and there are unprotected threads, use the value of THRDA plus one.
- When TWAIT=POOL, use THRDA plus *n* where *n* is the number of transactions that you want to be able to overflow to the pool.
- When TWAIT=NO, decide whether to allow more than the value in THRDA.

## Recommendations for accounting information for CICS threads

DB2 cuts accounting records when a thread signs on and when it terminates. CICS cuts accounting records at end-of-task. The CICS LU6.2 token gives you a way to correlate records between CICS and DB2.

***Using TOKENE to ensure proper accounting for tasks:*** Because it is possible for CICS tasks to reuse existing threads without signing on, one DB2 accounting record might contain data for several CICS tasks. If you specify YES in the DSNCRCT TYPE=ENTRY macro's TOKENE option, the CICS attachment facility passes the CICS LU6.2 token to DB2. It also forces DB2 to sign-on each new transaction so as to cut the accounting records. CICS generates an LU6.2 token for every CICS transaction, including both terminal and non-terminal-driven tasks.

Specify YES in the RCT TYPE=INIT macro's TOKENI option to set this default for all RCT entries.

The CICS accounting token is displayed on the DB2 PM Accounting Trace and the DB2 PM Online Monitor Thread Identification panel.

Specifying YES slightly increases the overhead of an SQL request that reuses threads, because of the additional sign-on activity.

# IMS design options

The IMS attachment facility gives you the following design options:

- Control the number of IMS regions connected to DB2. For IMS, this is also the maximum number of concurrent threads.
- Optimize the number of concurrent threads used by IMS.

  A dependent region with a subsystem member (SSM) that is not empty is connected to DB2 at start up time. Regions with a null SSM cannot create a thread to DB2. A thread to DB2 is created at the first execution of an SQL statement in an IMS application schedule; it is terminated when the application terminates.

  The maximum number of concurrent threads used by IMS can be controlled by the number of IMS regions that can connect to DB2 by transaction class assignments. You can control the number by doing the following:

  - Minimize the number of regions needing a thread by the way in which you assign applications to regions.
  - Provide an empty SSM member for regions that will not connect to DB2.
- Provide efficient thread reuse for high volume transactions.

Thread creation and termination is a significant cost in IMS transactions. IMS transactions identified as wait for input (WFI) can reuse threads: they create a thread at the first execution of an SQL statement and reuse it until the region is terminated. In general, though, use WFI only for transactions that reach a region utilization of at least 75%.

Some degree of thread reuse can also be achieved with IMS class scheduling, queuing, and a PROCLIM count greater than one. IMS Fast Path (IFP) dependent regions always reuse the DB2 thread.

## TSO design options

You can tune your TSO attachment facility by choosing values for the following parameters on the Storage Sizes installation panel (DSNTIPE):

**MAX TSO CONNECT** The maximum number of TSO foreground connections (including DB2I, QMF, and foreground applications)

**MAX BATCH CONNECT** The maximum number of TSO background connections (including batch jobs and utilities)

Because DB2 must be stopped to set new values, consider setting a higher MAX BATCH CONNECT for batch periods. The statistics record (IFCID 0001) provides information on the create thread queue. The DB2 PM statistics report (in Figure 73) shows that information under the SUBSYSTEM SERVICES section.

For TSO or batch environments, having 1% of the requests queued is probably a good number to aim for by adjusting the MAX USERS value of installation panel DSNTIPE. Queuing at create thread time is not desirable in the CICS and IMS environments. If you are running IMS or CICS in the same DB2 subsystem as TSO and batch, use MAX BATCH CONNECT and MAX TSO CONNECT to limit the number of threads taken by the TSO and batch environments. The goal is to allow enough threads for CICS and IMS so that their threads do not queue. To determine the number of allied threads queued, see the QUEUED AT CREATE THREAD field ( A ) of the DB2 PM statistics report.

```
SUBSYSTEM SERVICES          QUANTITY
--------------------------  --------
IDENTIFY                    30757.00
CREATE THREAD               30889.00
SIGNON                          0.00
TERMINATE                   61661.00
ROLLBACK                      644.00

COMMIT PHASE 1                  0.00
COMMIT PHASE 2                  0.00
READ ONLY COMMIT                0.00

UNITS OF RECOVERY INDOUBT       0.00
UNITS OF REC.INDBT RESOLVED     0.00

SYNCHS(SINGLE PHASE COMMIT) 30265.00
QUEUED AT CREATE THREAD  A      0.00
SUBSYSTEM ALLIED MEMORY EOT     1.00
SUBSYSTEM ALLIED MEMORY EOM     0.00
SYSTEM EVENT CHECKPOINT         0.00
```

*Figure 73. Thread queuing in the DB2 PM statistics report*

## QMF design options

QMF has the following significant performance options:

- The DSQSIROW parameter of the ISPSTART command
- SPACE parameter of the user QMF profile (Q.PROFILES)
- QMF region size and the spill file attributes
- TRACE parameter of the user QMF profile (Q.PROFILES)

For more information on these aspects of QMF and how they affect performance, see *Query Management Facility: Installing and Managing QMF on OS/390 and z/OS*.

# Chapter 30. Improving concurrency

This chapter begins with an overview of concurrency and locks in the following sections:
- “Definitions of concurrency and locks”,
- “Effects of DB2 locks” on page 644, and
- “Basic recommendations to promote concurrency” on page 646.

After the basic recommendations, the chapter tells what you can do about two major techniques that DB2 uses to control concurrency: *transaction locks* and *claims and drains*.

- **Transaction locks** mainly control access by SQL statements. Those locks are the ones over which you have the most control.
  - “Aspects of transaction locks” on page 650 describes the various types of transaction locks that DB2 uses and how they interact.
  - “Lock tuning” on page 664 describes what you can change to control locking. Your choices include:
    - “Startup procedure options” on page 665
    - “Installation options for wait times” on page 665
    - “Other options that affect locking” on page 670
    - “Bind options” on page 675
    - “Isolation overriding with SQL statements” on page 689
    - “The statement LOCK TABLE” on page 690

  Under those headings, *lock* (with no qualifier) refers to *transaction lock*.
- **Latches** are conceptually similar to locks in that they control serialization. They can improve concurrency because they are usually held for shorter duration than locks and they cannot “deadlatch”. However, page latches can wait, and this wait time is reported in accounting trace class 3. Because latches are not under your control, they are not described in any detail.
- **Claims and drains** provide another mechanism to control serialization. SQL applications and some utilities make claims for objects when they first access them. Operations that drain can take control of an object by quiescing the existing claimers and preventing new claims. After a drainer completes its operations, claimers can resume theirs. DB2 utilities, commands, and some SQL statements can act as drainers.

  “Claims and drains for concurrency control” on page 695 describes claims and drains in more detail and explains how to plan utility jobs and other activities to maximize efficiency.

The chapter ends by describing how you can monitor DB2’s use of locks and how you can analyze a sample problem in section “Monitoring of DB2 locking” on page 700.

DB2 extends its concurrency controls to multiple subsystems for data sharing. For information about that, see *DB2 Data Sharing: Planning and Administration*.

## Definitions of concurrency and locks

***Definition:*** *Concurrency* is the ability of more than one application process to access the same data at essentially the same time.

***Example:*** An application for order entry is used by many transactions simultaneously. Each transaction makes inserts in tables of invoices and invoice items, reads a table of data about customers, and reads and updates data about items on hand. Two operations on the same data, by two simultaneous transactions, might be separated only by microseconds. To the users, the operations appear concurrent.

***Conceptual background:*** Concurrency must be controlled to prevent lost updates and such possibly undesirable effects as unrepeatable reads and access to uncommitted data.

> **Lost updates.** Without concurrency control, two processes, A and B, might both read the same row from the database, and both calculate new values for one of its columns, based on what they read. If A updates the row with its new value, and then B updates the same row, A's update is lost.
>
> **Access to uncommitted data.** Also without concurrency control, process A might update a value in the database, and process B might read that value before it was committed. Then, if A's value is not later committed, but backed out, B's calculations are based on uncommitted (and presumably incorrect) data.
>
> **Unrepeatable reads.** Some processes require the following sequence of events: A reads a row from the database and then goes on to process other SQL requests. Later, A reads the first row again and must find the same values it read the first time. Without control, process B could have changed the row between the two read operations.

To prevent those situations from occurring unless they are specifically allowed, DB2 might use *locks* to control concurrency.

**What do locks do?** A lock associates a DB2 resource with an application process in a way that affects how other processes can access the same resource. The process associated with the resource is said to "hold" or "own" the lock. DB2 uses locks to ensure that no process accesses data that has been changed, but not yet committed, by another process.

**What do you do about locks?** To preserve data integrity, your application process acquires locks implicitly, that is, under DB2 control. It is not necessary for a process to request a lock explicitly to conceal uncommitted data. Therefore, sometimes you need not do anything about DB2 locks. Nevertheless processes acquire, or avoid acquiring, locks based on certain general parameters. You can make better use of your resources and improve concurrency by understanding the effects of those parameters.

## Effects of DB2 locks

The effects of locks that you want to minimize are *suspension*, *timeout*, and *deadlock*.

### Suspension

***Definition:*** An application process is *suspended* when it requests a lock that is already held by another application process and cannot be shared. The suspended process temporarily stops running.

***Order of precedence for lock requests:*** Incoming lock requests are queued. Requests for lock promotion, and requests for a lock by an application process that already holds a lock on the same object, precede requests for locks by new applications. Within those groups, the request order is "first in, first out".

***Example:*** Using an application for inventory control, two users attempt to reduce the quantity on hand of the same item at the same time. The two lock requests are queued. The second request in the queue is suspended and waits until the first request releases its lock.

***Effects:*** The suspended process resumes running when:

- All processes that hold the conflicting lock release it.
- The requesting process times out or deadlocks and the process resumes to deal with an error condition.

## Timeout

***Definition:*** An application process is said to *time out* when it is terminated because it has been suspended for longer than a preset interval.

***Example:*** An application process attempts to update a large table space that is being reorganized by the utility REORG TABLESPACE with SHRLEVEL NONE. It is likely that the utility job will not release control of the table space before the application process times out.

***Effects:*** DB2 terminates the process, issues two messages to the console, and returns SQLCODE -911 or -913 to the process (SQLSTATEs '40001' or '57033'). Reason code 00C9008E is returned in the SQLERRD(3) field of the SQLCA. If statistics trace class 3 is active, DB2 writes a trace record with IFCID 0196.

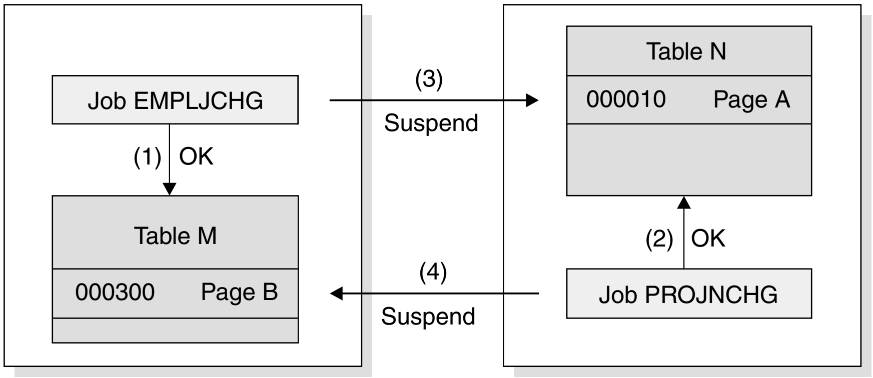
COMMIT and ROLLBACK operations do not time out. The command STOP DATABASE, however, may time out and send messages to the console, but it will retry up to 15 times.

**For more information** about setting the timeout interval, see “Installation options for wait times” on page 665.

## Deadlock

***Definition:*** A *deadlock* occurs when two or more application processes each hold locks on resources that the others need and without which they cannot proceed.

***Example:*** Figure 74 on page 646 illustrates a deadlock between two transactions.



**Notes:**

1. Jobs EMPLJCHG and PROJNCHG are two transactions. Job EMPLJCHG accesses table M, and acquires an exclusive lock for page B, which contains record 000300.
2. Job PROJNCHG accesses table N, and acquires an exclusive lock for page A, which contains record 000010.
3. Job EMPLJCHG requests a lock for page A of table N while still holding the lock on page B of table M. The job is suspended, because job PROJNCHG is holding an exclusive lock on page A.
4. Job PROJNCHG requests a lock for page B of table M while still holding the lock on page A of table N. The job is suspended, because job EMPLJCHG is holding an exclusive lock on page B. The situation is a deadlock.

*Figure 74. A deadlock example*

***Effects:*** After a preset time interval (the value of DEADLOCK TIME), DB2 can roll back the current unit of work for one of the processes or request a process to terminate. That frees the locks and allows the remaining processes to continue. If statistics trace class 3 is active, DB2 writes a trace record with IFCID 0172. Reason code 00C90088 is returned in the SQLERRD(3) field of the SQLCA. (The codes that describe DB2's exact response depend on the operating environment; for details, see Part 5 of *DB2 Application Programming and SQL Guide*.)

It is possible for two processes to be running on distributed DB2 subsystems, each trying to access a resource at the other location. In that case, neither subsystem can detect that the two processes are in deadlock; the situation resolves only when one process times out.

## Basic recommendations to promote concurrency

Recommendations are grouped roughly by their scope, as:
- "Recommendations for system options"
- "Recommendations for database design" on page 647
- "Recommendations for application design" on page 648

## Recommendations for system options

***Reduce swapping:*** If a task is waiting or is swapped out and the unit of work has not been committed, then it still holds locks. When a system is heavily loaded, contention for processing, I/O, and storage can cause waiting. Consider reducing the number of initiators, increasing the priority for the DB2 tasks, and providing more processing, I/O, or storage resources.

***Make way for the IRLM:*** Make sure that the IRLM has a high MVS dispatching priority or is assigned to the SYSSTC service class. It should come next after VTAM and before DB2.

If you can define more ECSA, then start the IRLM with PC=NO rather than PC=YES. You can make this change without changing your application process. This change can also reduce processing time.

***Restrict updating of partitioning key columns:*** In systems with high concurrency and long running transactions, allowing updating of partitioning key columns when the update moves the row from one partition to another can cause concurrency problems. Allow updating only when the row stays in the same partition by setting the UPDATE PART KEY COLS field in DSNTIP4 to SAME.

## Recommendations for database design

***Keep like things together:*** Cluster tables relevant to the same application into the same database, and give each application process that creates private tables a private database in which to do it. In the ideal model, each application process uses as few databases as possible.

***Keep unlike things apart:*** Give users different authorization IDs for work with different databases; for example, one ID for work with a shared database and another for work with a private database. This effectively adds to the number of possible (but not concurrent) application processes while minimizing the number of databases each application process can access.

***Plan for batch inserts:*** If your application does sequential batch insertions, excessive contention on the space map pages for the table space can occur. This problem is especially apparent in data sharing, where contention on the space map means the added overhead of page P-lock negotiation. For these types of applications, consider using the MEMBER CLUSTER option of CREATE TABLESPACE. This option causes DB2 to disregard the clustering index (or implicit clustering index) when assigning space for the SQL INSERT statement. For more information about using this option in data sharing, see Chapter 6 of *DB2 Data Sharing: Planning and Administration*. For the syntax, see Chapter 5 of *DB2 SQL Reference*.

***Use LOCKSIZE ANY until you have reason not to:*** LOCKSIZE ANY is the default for CREATE TABLESPACE. It allows DB2 to choose the lock size, and DB2 usually chooses LOCKSIZE PAGE and LOCKMAX SYSTEM for non-LOB table spaces. For LOB table spaces, it chooses LOCKSIZE LOB and LOCKMAX SYSTEM. You should use LOCKSIZE TABLESPACE or LOCKSIZE TABLE only for read-only table spaces or tables, or when concurrent access to the object is not needed. Before you choose LOCKSIZE ROW, you should estimate whether there will be an increase in overhead for locking and weigh that against the increase in concurrency.

***Examine small tables:*** For small tables with high concurrency requirements, estimate the number of pages in the data and in the index. If the index entries are short or they have many duplicates, then the entire index can be one root page and a few leaf pages. In this case, spread out your data to improve concurrency, or consider it a reason to use row locks.

***Partition the data:*** Online queries typically make few data changes, but they occur often. Batch jobs are just the opposite; they run for a long time and change many rows, but occur infrequently. The two do not run well together. You might be able to separate online applications from batch, or two batch jobs from each other. To separate online and batch applications, provide separate partitions. Partitioning can also effectively separate batch jobs from each other.

***Fewer rows of data per page:*** By using the MAXROWS clause of CREATE or ALTER TABLESPACE, you can specify the maximum number of rows that can be on a page. For example, if you use MAXROWS 1, each row occupies a whole page, and you confine a page lock to a single row. Consider this option if you have a reason to avoid using row locking, such as in a data sharing environment where row locking overhead can be excessive.

## Recommendations for application design

***Access data in a consistent order:*** When different applications access the same data, try to make them do so in the same sequence. For example, make both access rows 1,2,3,5 in that order. In that case, the first application to access the data delays the second, but the two applications cannot deadlock. For the same reason, try to make different applications access the same tables in the same order.

***Commit work as soon as is practical:*** To avoid unnecessary lock contentions, issue a COMMIT statement as soon as possible after reaching a point of consistency, even in read-only applications. To prevent unsuccessful SQL statements (such as PREPARE) from holding locks, issue a ROLLBACK statement after a failure. Statements issued through SPUFI can be committed immediately by the SPUFI autocommit feature.

Taking commit points frequently in a long running unit of recovery (UR) has the following benefits:

- Reduces lock contention
- Improves the effectiveness of lock avoidance, especially in a data sharing environment
- Reduces the elapsed time for DB2 system restart following a system failure
- Reduces the elapsed time for a unit of recovery to rollback following an application failure or an explicit rollback request by the application
- Provides more opportunity for utilities, such as online REORG, to break in

Consider using the UR CHECK FREQ field or the UR LOG WRITE CHECK field of installation panel DSNTIPN to help you identify those applications that are not committing frequently. UR CHECK FREQ, which identifies when too many checkpoints have occurred without a UR issuing a commit, is helpful in monitoring overall system activity. UR LOG WRITE CHECK enables you to detect applications that might write too many log records between commit points, potentially creating a lengthy recovery situation for critical tables.

Even though an application might conform to the commit frequency standards of the installation under normal operational conditions, variation can occur based on system workload fluctuations. For example, a low-priority application might issue a commit frequently on a system that is lightly loaded. However, under a heavy system load, the use of the CPU by the application may be pre-empted, and, as a result, the application may violate the rule set by the UR CHECK FREQ parameter. For this reason, add logic to your application to commit based on time elapsed since last commit, and not solely based on the amount of SQL processing performed. In addition, take frequent commit points in a long running unit of work that is read-only to reduce lock contention and to provide opportunities for utilities, such as online REORG, to access the data.

***Retry an application after deadlock or timeout:*** Include logic in a batch program so that it retries an operation after a deadlock or timeout. Such a method could help you recover from the situation without assistance from operations personnel. Field SQLERRD(3) in the SQLCA returns a reason code that indicates whether a deadlock or timeout occurred.

***Close cursors:*** If you define a cursor using the WITH HOLD option, the locks it needs can be held past a commit point. Use the CLOSE CURSOR statement as soon as possible in your program to cause those locks to be released and the resources they hold to be freed at the first commit point that follows the CLOSE CURSOR statement. Whether page or row locks are held for WITH HOLD cursors is controlled by the RELEASE LOCKS parameter on panel DSNTIP4.

***Free locators:*** If you have executed, the HOLD LOCATOR statement, the LOB locator holds locks on LOBs past commit points. Use the FREE LOCATOR statement to release these locks.

***Bind plans with ACQUIRE(USE):*** ACQUIRE(USE), which indicates that DB2 will acquire table and table space locks when the objects are first used and not when the plan is allocated, is the best choice for concurrency. Packages are always bound with ACQUIRE(USE), by default. ACQUIRE(ALLOCATE) can provide better protection against timeouts. Consider ACQUIRE(ALLOCATE) for applications that need gross locks instead of intent locks or that run with other applications that may request gross locks instead of intent locks. Acquiring the locks at plan allocation also prevents any one transaction in the application from incurring the cost of acquiring the table and table space locks. If you need ACQUIRE(ALLOCATE), you might want to bind all DBRMs directly to the plan.

***Bind with ISOLATION(CS) and CURRENTDATA(NO) typically:*** ISOLATION(CS) lets DB2 release acquired row and page locks as soon as possible. CURRENTDATA(NO) lets DB2 avoid acquiring row and page locks as often as possible. After that, in order of decreasing preference for concurrency, use these bind options:

1. ISOLATION(CS) with CURRENTDATA(YES), when data returned to the application must not be changed before your next FETCH operation.
2. ISOLATION(RS), when data returned to the application must not be changed before your application commits or rolls back. However, you do not care if other application processes insert additional rows.
3. ISOLATION(RR), when data evaluated as the result of a query must not be changed before your application commits or rolls back. New rows cannot be inserted into the answer set.

For updatable scrollable cursors, ISOLATION(CS) provides the additional advantage of letting DB2 use *optimistic concurrency control* to further reduce the amount of time that locks are held. For more information about optimistic concurrency control, see “Advantages and disadvantages of the isolation values” on page 680.

***Use ISOLATION(UR) cautiously:*** UR isolation acquires almost no locks on rows or pages. It is fast and causes little contention, but it reads uncommitted data. Do not use it unless you are sure that your applications and end users can accept the logical inconsistencies that can occur.

***Use global transactions:***The Recoverable Resource Manager Services attachment facility (RRSAF) relies on an OS/390 component called OS/390 Transaction Management and Recoverable Resource Manager Services (OS/390 RRS). OS/390

RRS provides system-wide services for coordinating two-phase commit operations across MVS products. For RRSAF applications and IMS transactions that run under OS/390 RRS, you can group together a number of DB2 agents into a single global transaction. A global transaction allows multiple DB2 agents to participate in a single global transaction and thus share the same locks and access the same data. When two agents that are in a global transaction access the same DB2 object within a unit of work, those agents will not deadlock with each other. The following restrictions apply:

- There is no Parallel Sysplex support for global transactions.
- Because each of the ″branches″ of a global transaction are sharing locks, uncommitted updates issued by one branch of the transaction are visible to other branches of the transaction.
- Claim/drain processing is not supported across the branches of a global transaction, which means that attempts to issue CREATE, DROP, ALTER, GRANT, or REVOKE may deadlock or timeout if they are requested from different branches of the same global transaction.
- Attempts to update a partitioning key may deadlock or timeout because of the same restrictions on claim/drain processing.
- LOCK TABLE may deadlock or timeout across the branches of a global transaction.

For information on how to make an agent part of a global transaction for RRSAF applications, see Section 7 of *DB2 Application Programming and SQL Guide*.

## Aspects of transaction locks

Transaction locks have the following four basic aspects:

- “The size of a lock”
- “The duration of a lock” on page 654
- “The mode of a lock” on page 654
- “The object of a lock” on page 656

Knowing the aspects helps you understand why a process suspends or times out or why two processes deadlock. To change the situation, you also need to know:

- “DB2’s choice of lock types” on page 659

## The size of a lock

### Definition

The *size* (sometimes *scope* or *level*) of a lock on data in a table describes the amount of data controlled. The possible sizes of locks are table space, table, partition, page, and row. This section contains information about locking for non-LOB data. See “LOB locks” on page 691 for information on locking for LOBs.

### Hierarchy of lock sizes

The same piece of data can be controlled by locks of different sizes. A table space lock (the largest size) controls the most data, all the data in an entire table space. A page or row lock controls only the data in a single page or row.

As Figure 75 on page 651 suggests, row locks and page locks occupy an equal place in the hierarchy of lock sizes.

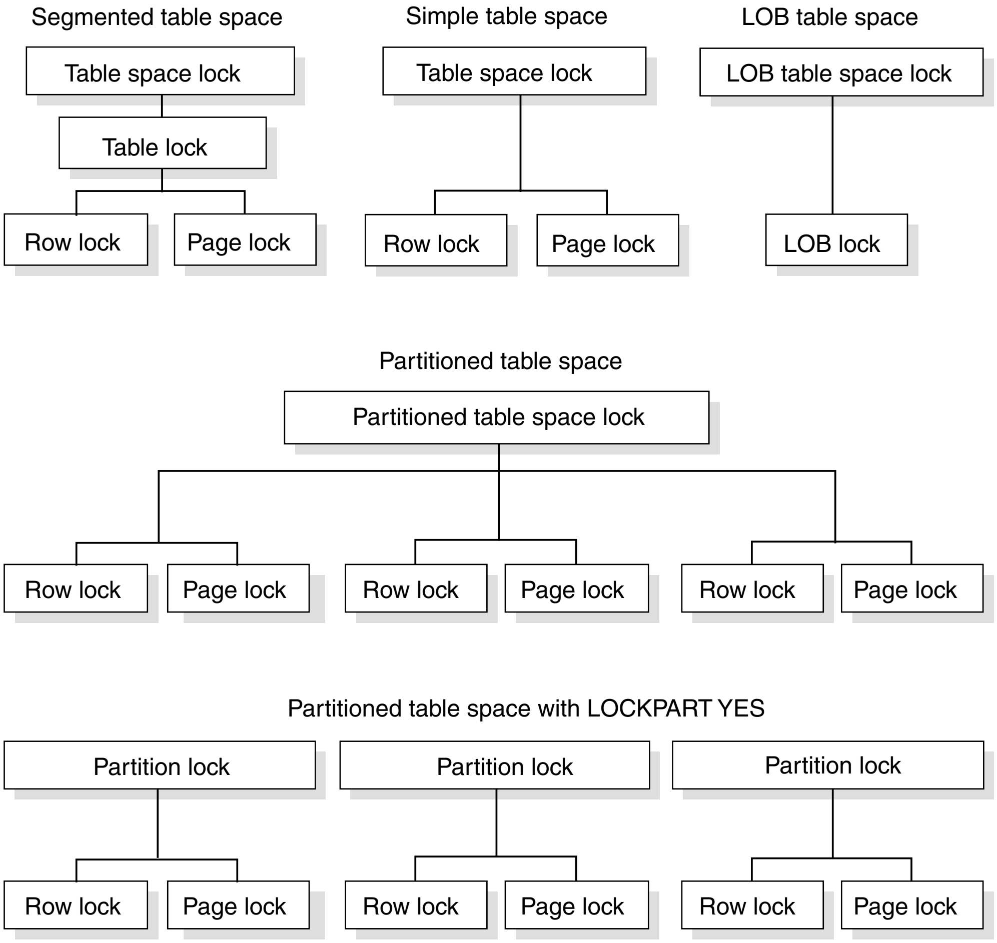


*Figure 75. Sizes of objects locked*

## General effects of size

Locking larger or smaller amounts of data allows you to trade performance for concurrency. Using page or row locks instead of table or table space locks has the following effects:

- Concurrency usually improves, meaning better response times and higher throughput rates for many users.
- Processing time and use of storage increases. That is especially evident in batch processes that scan or update a large number of rows.

Using only table or table space locks has the following effects:

- Processing time and storage usage is reduced.
- Concurrency can be reduced, meaning longer response times for some users but better throughput for one user.

## Effects of table spaces of different types

- The LOCKPART clause of CREATE and ALTER TABLESPACE lets you control how DB2 locks **partitioned table spaces**. The default, LOCKPART NO, means that one lock is used to lock the entire partitioned table space when any partition is accessed. LOCKPART NO is the value you want in most cases.

  With LOCKPART YES, individual partitions are locked only as they are accessed.

One case for using LOCKPART YES is for some data sharing applications, as described in Chapter 6 of *DB2 Data Sharing: Planning and Administration*. There are also benefits to non-data-sharing applications that use partitioned table spaces. For these applications, it might be desirable to acquire gross locks (S, U, or X) on partitions to avoid numerous lower level locks and yet still maintain concurrency. When locks escalate and the table space is defined with LOCKPART YES, applications that access different partitions of the same table space do not conflict during update activity.

***Restrictions:*** If any of the following conditions are true, DB2 must lock *all* partitions when LOCKPART YES is used:

– The plan is bound with ACQUIRE(ALLOCATE).
– The table space is defined with LOCKSIZE TABLESPACE.
– LOCK TABLE IN EXCLUSIVE MODE or LOCK TABLE IN SHARE MODE is used (without the PART option).

No matter how LOCKPART is defined, utility jobs can control separate partitions of a table space or index space and can run concurrently with operations on other partitions.

- A **simple table space** can contain more than one table. A lock on the table space locks all the data in every table. A single page of the table space can contain rows from every table. A lock on a page locks every row in the page, no matter what tables the data belongs to. Thus, a lock needed to access data from one table can make data from other tables temporarily unavailable. That effect can be partly undone by using row locks instead of page locks. But that step does not relieve the sweeping effect of a table space lock.
- In a **segmented table space**, rows from different tables are contained in different pages. Locking a page does not lock data from more than one table. Also, DB2 can acquire a table lock, which locks only the data from one specific table. Because a single row, of course, contains data from only one table, the effect of a row lock is the same as for a simple or partitioned table space: it locks one row of data from one table.
- In a **LOB table space**, pages are not locked. Because there is no concept of a row in a LOB table space, rows are not locked. Instead, LOBs are locked. See “LOB locks” on page 691 for more information.

## Differences between simple and segmented table spaces

Figure 76 on page 653 illustrates the difference between the effects of page locks on simple and segmented table spaces. Suppose that tables T1 and T2 reside in table space TS1. In a *simple* table space, a single page can contain rows from both T1 and T2. If User 1 and User 2 acquire incompatible locks on different pages, such as exclusive locks for updating data, neither can access all the rows in T1 and T2 until one of the locks is released. (User 1 and User 2 can both hold a page lock on the same page when the mode of the locks are compatible, such as locks for reading data.)

As the figure also shows, in a *segmented* table space, a table lock applies only to segments assigned to a single table. Thus, User 1 can lock all pages assigned to the segments of T1 while User 2 locks all pages assigned to segments of T2. Similarly, User 1 can lock a page of T1 without locking any data in T2.

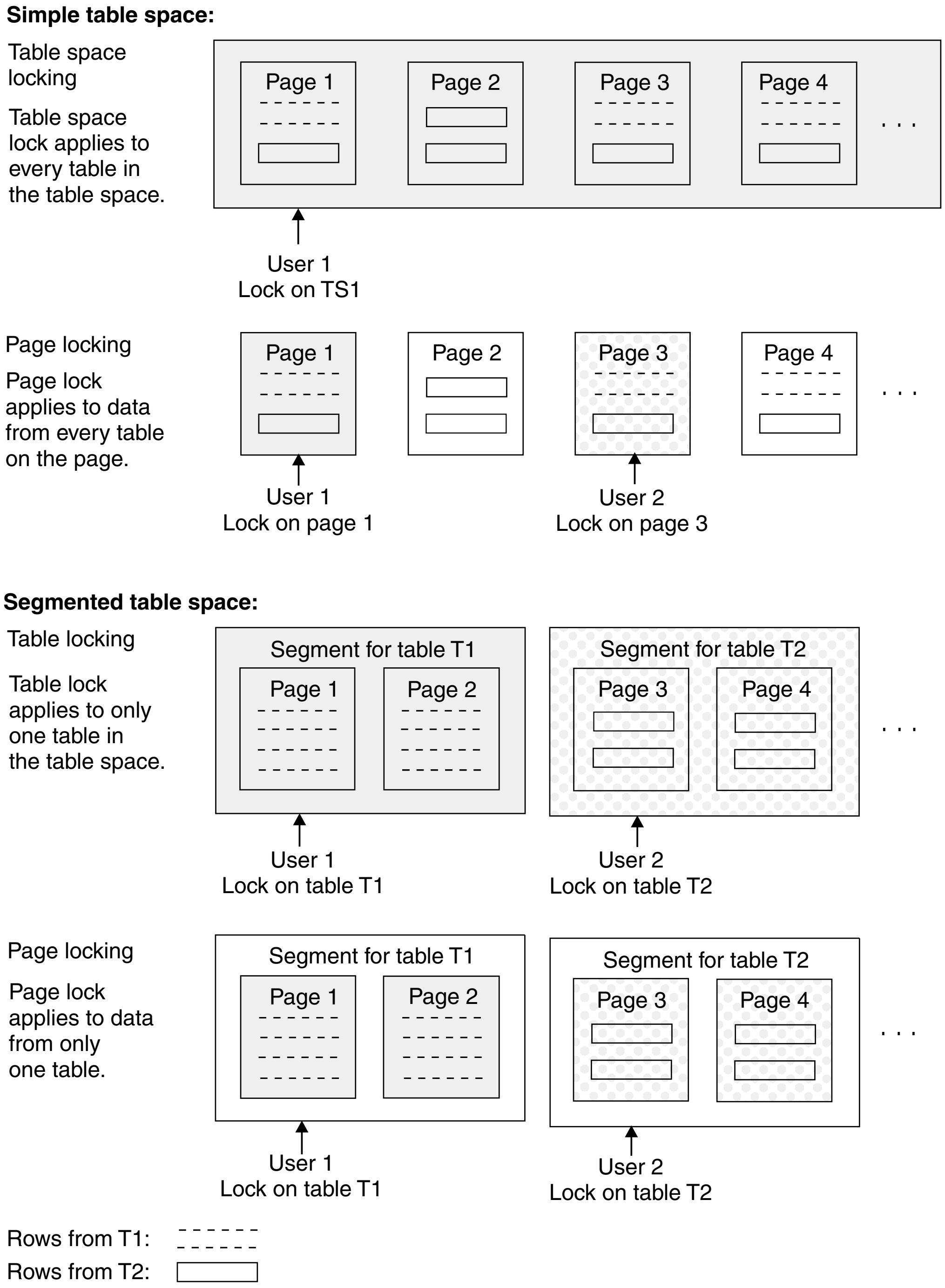


*Figure 76. Page locking for simple and segmented table spaces*

For information about controlling the size of locks, see:

- “LOCKSIZE clause of CREATE and ALTER TABLESPACE” on page 671
- “The statement LOCK TABLE” on page 690

## The duration of a lock

### Definition

The *duration* of a lock is the length of time the lock is held. It varies according to when the lock is acquired and when it is released.

### Effects

For maximum concurrency, locks on a small amount of data held for a short duration are better than locks on a large amount of data held for a long duration. However, acquiring a lock requires processor time, and holding a lock requires storage; thus, acquiring and holding one table space lock is more economical than acquiring and holding many page locks. Consider that trade-off to meet your performance and concurrency objectives.

***Duration of partition, table, and table space locks:*** Partition, table, and table space locks can be acquired when a plan is first allocated, or you can delay acquiring them until the resource they lock is first used. They can be released at the next commit point or be held until the program terminates.

On the other hand, LOB table space locks are always acquired when needed and released at a commit or held until the program terminates. See “LOB locks” on page 691 for information about locking LOBs and LOB table spaces.

***Duration of page and row locks:*** If a page or row is locked, DB2 acquires the lock only when it is needed. When the lock is released depends on many factors, but it is rarely held beyond the next commit point.

For information about controlling the duration of locks, see “Bind options” on page 675.

## The mode of a lock

### Definition

The *mode* (sometimes *state*) of a lock tells what access to the locked object is permitted to the lock owner and to any concurrent processes.

The possible modes for page and row locks and the modes for partition, table, and table space locks are listed below. See “LOB locks” on page 691 for more information about modes for LOB locks and locks on LOB table spaces.

When a page or row is locked, the table, partition, or table space containing it is also locked. In that case, the table, partition, or table space lock has one of the *intent* modes: IS, IX, or SIX. The modes S, U, and X of table, partition, and table space locks are sometimes called *gross* modes. In the context of reading, SIX is a gross mode lock because you don’t get page or row locks; in this sense, it is like an S lock.

***Example:*** An SQL statement locates John Smith in a table of customer data and changes his address. The statement locks the entire table space in mode IX and the specific row that it changes in mode X.

### Modes of page and row locks

Modes and their effects are listed in the order of increasing control over resources.

**S (SHARE)** The lock owner and any concurrent processes can read, but not change, the locked page or row. Concurrent processes can acquire S or U locks on the page or row or might read data without acquiring a page or row lock.

**U (UPDATE)** The lock owner can read, but not change, the locked page or row. Concurrent processes can acquire S locks or might read data without acquiring a page or row lock, but no concurrent process can acquire a U lock.

U locks reduce the chance of deadlocks when the lock owner is reading a page or row to determine whether to change it, because the owner can start with the U lock and then promote the lock to an X lock to change the page or row.

**X (EXCLUSIVE)**
The lock owner can read or change the locked page or row. A concurrent process can access the data if the process runs with UR isolation. (A concurrent process that is bound with cursor stability and CURRENTDATA(NO) can also read X-locked data if DB2 can tell that the data is committed.)

## Modes of table, partition, and table space locks

Modes and their effects are listed in the order of increasing control over resources.

**IS (INTENT SHARE)** The lock owner can read data in the table, partition, or table space, but not change it. Concurrent processes can both read and change the data. The lock owner might acquire a page or row lock on any data it reads.

**IX (INTENT EXCLUSIVE)** The lock owner and concurrent processes can read and change data in the table, partition, or table space. The lock owner might acquire a page or row lock on any data it reads; it must acquire one on any data it changes.

**S (SHARE)** The lock owner and any concurrent processes can read, but not change, data in the table, partition, or table space. The lock owner does not need page or row locks on data it reads.

**U (UPDATE)** The lock owner can read, but not change, the locked data; however, the owner can promote the lock to an X lock and then can change the data. Processes concurrent with the U lock can acquire S locks and read the data, but no concurrent process can acquire a U lock. The lock owner does not need page or row locks.

U locks reduce the chance of deadlocks when the lock owner is reading data to determine whether to change it. U locks are acquired on a table space when locksize is TABLESPACE and the statement is SELECT FOR UPDATE OF. Similarly, U locks are acquired on a table when lock size is TABLE and the statement is SELECT FOR UPDATE OF.

**SIX (SHARE with INTENT EXCLUSIVE)**
The lock owner can read and change data in the table, partition, or table space. Concurrent processes can read data in the table, partition, or

table space, but not change it. Only when the lock owner changes data does it acquire page or row locks.

**X (EXCLUSIVE)**
The lock owner can read or change data in the table, partition, or table space. A concurrent process can access the data if the process runs with UR isolation or if data in a LOCKPART(YES) table space is running with CS isolation and CURRENTDATA(NO). The lock owner does not need page or row locks.

### Lock mode compatibility

The major effect of the lock mode is to determine whether one lock is compatible with another.

***Definition:*** Locks of some modes do not shut out all other users. Assume that application process A holds a lock on a table space that process B also wants to access. DB2 requests, on behalf of B, a lock of some particular mode. If the mode of A's lock permits B's request, the two locks (or modes) are said to be *compatible*.

***Effects of incompatibility:*** If the two locks are not compatible, B cannot proceed. It must wait until A releases its lock. (And, in fact, it must wait until all existing incompatible locks are released.)

***Compatible lock modes:*** Compatibility for page and row locks is easy to define. Table 89 shows whether page locks of any two modes, or row locks of any two modes, are compatible (Yes) or not (No). No question of compatibility of a page lock with a row lock can arise, because a table space cannot use both page and row locks.

*Table 89. Compatibility of page lock and row lock modes*

| Lock Mode | S | U | X |
|---|---|---|---|
| S | Yes | Yes | No |
| U | Yes | No | No |
| X | No | No | No |

Compatibility for table space locks is slightly more complex. Table 90 shows whether or not table space locks of any two modes are compatible.

*Table 90. Compatibility of table and table space (or partition) lock modes*

| Lock Mode | IS | IX | S | U | SIX | X |
|---|---|---|---|---|---|---|
| IS | Yes | Yes | Yes | Yes | Yes | No |
| IX | Yes | Yes | No | No | No | No |
| S | Yes | No | Yes | Yes | No | No |
| U | Yes | No | Yes | No | No | No |
| SIX | Yes | No | No | No | No | No |
| X | No | No | No | No | No | No |

## The object of a lock

### Definition and examples

The *object* of a lock is the resource being locked.

You might have to consider locks on any of the following objects:

- **User data in target tables**. A *target table* is a table that is accessed specifically in an SQL statement, either by name or through a view. Locks on those tables are the most common concern, and the ones over which you have most control.
- **User data in related tables**. Operations subject to referential constraints can require locks on related tables. For example, if you delete from a parent table, DB2 might delete rows from the dependent table as well. In that case, DB2 locks data in the dependent table as well as in the parent table.

  Similarly, operations on rows that contain LOB values might require locks on the LOB table space and possibly on LOB values within that table space. See “LOB locks” on page 691 for more information.

  If your application uses triggers, any triggered SQL statements can cause additional locks to be acquired.
- **DB2 internal objects**. Most of these you are never aware of, but you might notice the following locks on internal objects:
  - Portions of the **DB2 catalog**. For more information, see “Locks on the DB2 catalog”.
  - The **skeleton cursor table** (SKCT) representing an application plan.
  - The **skeleton package table** (SKPT) representing a package. For more information on skeleton tables, see “Locks on the skeleton tables (SKCT and SKPT)” on page 658.
  - The **database descriptor** (DBD) representing a DB2 database. For more information, see “Locks on the database descriptors (DBDs)” on page 658.

## Indexes and data-only locking

No index page locks are acquired during processing. Instead, DB2 uses a technique called *data-only locking* to serialize changes. Index page latches are acquired to serialize changes within a page and guarantee that the page is physically consistent. Acquiring page latches ensures that transactions accessing the same index page concurrently do not see the page in a partially changed state.

The underlying data page or row locks are acquired to serialize the reading and updating of index entries to ensure the data is logically consistent, meaning that the data is committed and not subject to rollback or abort. The data locks can be held for a long duration such as until commit. However, the page latches are only held for a short duration while the transaction is accessing the page. Because the index pages are not locked, hot spot insert scenarios (which involve several transactions trying to insert different entries into the same index page at the same time) do not cause contention problems in the index.

A query that uses index-only access might lock the data page or row, and that lock can contend with other processes that lock the data. However, using lock avoidance techniques can reduce the contention. See “Lock avoidance” on page 686 for more information about lock avoidance.

## Locks on the DB2 catalog

SQL data definition statements, GRANT statements, and REVOKE statements require locks on the DB2 catalog. If different application processes are issuing these types of statements, catalog contention can occur. You can take action to avoid contention.

***Contention within table space SYSDBASE:*** SQL statements that update the catalog table space SYSDBASE contend with each other when those statements are on the same table space. Those statements are:

CREATE, ALTER, and DROP TABLESPACE, TABLE, and INDEX
CREATE and DROP VIEW, SYNONYM, and ALIAS

COMMENT ON and LABEL ON
GRANT and REVOKE of table privileges

***Recommendation:*** Reduce the concurrent use of statements that update SYSDBASE for the same table space.

***Contention independent of databases:*** The following limitations on concurrency are independent of the referenced database:

- CREATE and DROP statements for a table space or index that uses a storage group contend significantly with other such statements.
- CREATE, ALTER, and DROP DATABASE, and GRANT and REVOKE database privileges all contend with each other and with any other function that requires a database privilege.
- CREATE, ALTER, and DROP STOGROUP contend with any SQL statements that refer to a storage group and with extensions to table spaces and indexes that use a storage group.
- GRANT and REVOKE for plan, package, system, or use privileges contend with other GRANT and REVOKE statements for the same type of privilege and with data definition statements that require the same type of privilege.

## Locks on the skeleton tables (SKCT and SKPT)

The SKCT of a plan, or the SKPT of a package, is locked while the plan or package is running. The following operations require incompatible locks on the SKCT or SKPT, whichever is applicable, and cannot run concurrently:

- Binding, rebinding, or freeing the plan or package
- Dropping a resource or revoking a privilege that the plan or package depends on
- In some cases, altering a resource that the plan or package depends on

## Locks on the database descriptors (DBDs)

Whether a process locks a target DBD depends largely on whether the DBD is already in the EDM pool.

**If the DBD is not in the EDM pool**, most processes acquire locks on the database descriptor table space (DBD01). That has the effect of locking the DBD and can cause conflict with other processes.

**If the DBD is in the EDM pool**, the lock on the DBD depends on the type of process, as shown in Table 91.

*Table 91. Contention for locks on a DBD in the EDM pool*

| Process Type | Process | Lock acquired | Conflicts with process type |
|---|---|---|---|
| 1 | Static DML statements (SELECT, DELETE, INSERT, UPDATE) | None | None |
| **Note:** Static DML statements can conflict with other processes because of locks on data. | | | |
| 2 | Dynamic DML statements | S | 3 |
| **Note:** If caching of dynamic SQL is turned on, no lock is taken on the DBD when a statement is prepared for insertion in the cache or for a statement in the cache. | | | |
| 3 | Data definition statements (ALTER, CREATE, DROP) | X | 2,3,4 |
| 4 | Utilities | S | 3 |

# DB2’s choice of lock types

***Overview:*** For the locks acquired on target tables by different types of SQL data manipulation statements, see:

- “Modes of locks acquired for SQL statements”

The lock acquired because of an SQL statement is not always a constant throughout the time of execution. For two situations in which DB2 can change acquired locks during execution, see:

- “Lock promotion” on page 662
- “Lock escalation” on page 662

For a summary of the locks acquired by other operations, see:

- “Modes of transaction locks for various processes” on page 664

## Modes of locks acquired for SQL statements

Table 92 on page 660 shows the modes of locks that a process acquires. The mode depends on:

- The type of processing being done
- The value of LOCKSIZE for the target table
- The value of ISOLATION with which the plan or package is bound
- The method of access to data

For details about:

- LOCKSIZE, see “LOCKSIZE clause of CREATE and ALTER TABLESPACE” on page 671
- ISOLATION, see “The ISOLATION option” on page 678
- Access methods, see “Chapter 33. Using EXPLAIN to improve SQL performance” on page 789

***Reading the table:*** The following SQL statements and sample steps provide a way to understand the table that shows the modes of locks.

```
EXEC SQL DELETE FROM DSN8710.EMP WHERE CURRENT OF C1;
```

Use the following sample steps to understand the table:

1. Find the section of the table for DELETE operations using a cursor. It is on page 661.
2. Find the row for the appropriate values of LOCKSIZE and ISOLATION. Table space DSN8710 is defined with LOCKSIZE ANY. If the value of ISOLATION was not specifically chosen, it is RR by default.
3. Find the subrow for the expected access method. The operation probably uses the index on employee number. Because the operation deletes a row, it must update the index. Hence, you can read the locks acquired in the subrow for “Index, updated”:
   - An IX lock on the table space
   - An IX lock on the table (but see the step that follows)
   - An X lock on the page containing the row that is deleted
4. Check the notes to the entries you use, at the end of the table. For this sample operation, see:
   - Note 2, on the column heading for “Table”. If the table is not segmented, there is no separate lock on the table.
   - Note 3, on the column heading for “Data Page or Row”. Because LOCKSIZE for the table space is ANY, DB2 can choose whether to use page locks, table locks, or table space locks. Typically it chooses page locks.

*Table 92. Modes of locks acquired for SQL statements. Numbers in parentheses () refer to numbered notes beginning on page 661.*

| | | | Lock Mode | | |
|---|---|---|---|---|---|
| **LOCKSIZE** | **ISOLATION** | **Access Method (1)** | **Table space (10)** | **Table (2)** | **Data page or row (3)** |
| **Processing statement: SELECT with read-only or ambiguous cursor, or with no cursor. UR isolation is allowed and requires none of these locks.** | | | | | |
| TABLESPACE | CS RS RR | Any | S | n/a | n/a |
| TABLE (2) | CS RS RR | Any | IS | S | n/a |
| PAGE, ROW, or ANY | CS or RS | Index, any use | IS(4) (11) | IS(4) | S(5) |
| | | Table space scan | IS(4) (11) | IS(4) | S(5) |
| PAGE, ROW, or ANY | RR | Index/data probe | IS(4) | IS(4) | S |
| | | Index scan (6) | IS(4) or S | S, IS(4), or n/a | S or n/a |
| | | Table space scan (6) | IS(2) or S | S or n/a | n/a |
| **Processing statement: INSERT ... VALUES(...) or INSERT ... fullselect (7)** | | | | | |
| TABLESPACE | CS RS RR | Any | X | n/a | n/a |
| TABLE (2) | CS RS RR | Any | IX | X | n/a |
| PAGE, ROW, or ANY | CS RS RR | Any | IX | IX | X |
| **Processing statement: UPDATE or DELETE, without cursor. Data page and row locks apply only to selected data.** | | | | | |
| TABLESPACE | CS RS RR | Any | X | n/a | n/a |
| TABLE (2) | CS RS RR | Any | IX | X | n/a |
| PAGE, ROW, or ANY | CS | Index selection | IX | IX | For delete: X. For update: U→X. |
| | | Index/data selection | IX | IX | U→X |
| | | Table space scan | IX | IX | U→X |
| PAGE, ROW, or ANY | RS | Index selection | IX | IX | For update: S or U(9)→X. For delete: [S→X] or X. |
| | | Index/data selection | IX | IX | S or U(9)→X |
| | | Table space scan | IX | IX | S or U(9)→X |
| PAGE, ROW, or ANY | RR | Index selection | IX | IX | For update: [S or U(9)→X] or X. For delete: [S→X] or X. |
| | | Index/data selection | IX | IX | S or U(9)→X |
| | | Table space scan | IX(2) or X | X or n/a | n/a |
| **Processing Statement: SELECT with FOR UPDATE OF. Data page and row locks apply only to selected data.** | | | | | |
| TABLESPACE | CS RS RR | Any | U | n/a | n/a |
| TABLE (2) | CS RS RR | Any | IS or IX | U | n/a |
| PAGE, ROW, or ANY | CS | Index, any use | IX | IX | U |
| | | Table space scan | IX | IX | U |
| PAGE, ROW, or ANY | RS | Index, any use | IX | IX | S, U, or X(9) |
| | | Table space scan | IX | IX | S, U, or X(9) |

*Table 92. Modes of locks acquired for SQL statements (continued). Numbers in parentheses () refer to numbered notes beginning on page 661.*

| | | | Lock Mode | | |
|---|---|---|---|---|---|
| **LOCKSIZE** | **ISOLATION** | **Access Method (1)** | **Table space (10)** | **Table (2)** | **Data page or row (3)** |
| PAGE, ROW, or ANY | RR | Index/data probe | IX | IX | S, U, or X(9) |
| | | Index scan (6) | IX or X | X, IX, or n/a | S, U, X(9), or n/a |
| | | Table space scan (6) | IX(2) or X | X or n/a | S, U, X(9), or n/a |
| **Processing Statement:** | | **UPDATE or DELETE with cursor** | | | |
| TABLESPACE | Any | Any | X | n/a | n/a |
| TABLE (2) | Any | Any | IX | X | n/a |
| PAGE, ROW, or ANY | CS, RS, or RR | Index, updated | IX | IX | X |
| | | Index not updated | IX | IX | X |

## Notes for Table 92 on page 660

1. All access methods are either scan-based or probe-based. Scan-based means the index or table space is scanned for successive entries or rows. Probe-based means the index is searched for an entry as opposed to a range of entries, which a scan does. ROWIDs provide data probes to look for a single data row directly. The type of lock used depends on the backup access method. Access methods may be index-only, data-only, or index-to-data.

   **Index-only**
   The index alone identifies qualifying rows and the return data.

   **Data-only:**
   The data alone identifies qualifying rows and the return data, such as a table space scan or the use of ROWID for a probe.

   **Index-to-data**
   The index is used or the index plus data are used to evaluate the predicate:

   - **Index selection**: index is used to evaluate predicate and data is used to return values.
   - **Index/data selection**: index and data are used to evaluate predicate and data is used to return values.
2. Used for segmented table spaces only.
3. These locks are taken on pages if LOCKSIZE is PAGE or on rows if LOCKSIZE is ROW. When the maximum number of locks per table space (LOCKMAX) is reached, locks escalate to a table lock for tables in a segmented table space, or to a table space lock for tables in a non-segmented table space. Using LOCKMAX 0 in CREATE or ALTER TABLESPACE disables lock escalation.
4. If the table or table space is started for read-only access, DB2 attempts to acquire an S lock. If an incompatible lock already exists, DB2 acquires the IS lock.
5. SELECT statements that do not use a cursor, or that use read-only or ambiguous cursors and are bound with CURRENTDATA(NO), might not require any lock if DB2 can determine that the data to be read is committed. This is known as *lock avoidance*.

6. Even if LOCKMAX is 0, the bind process can promote the lock size to TABLE or TABLESPACE. If that occurs, SQLCODE +806 is issued.
7. The locks listed are acquired on the object into which the insert is made. A subselect acquires additional locks on the objects it reads, as if for SELECT with read-only cursor or ambiguous cursor, or with no cursor.
8. The U lock is taken if index columns are updated.
9. Whether the lock is S or U is determined by an installation option. For a full description, see “The option U LOCK FOR RR/RS” on page 673. If you use the WITH clause to specify the isolation as RR or RS, you can use the KEEP UPDATE LOCKS option to obtain and hold an X lock instead of a U or S lock.
10. Includes partition locks, if selective partition locking is used. Does not include LOB table space locks. See “LOB locks” on page 691 for information about locking LOB table spaces.
11. If the table space is defined with LOCKPART YES, it is possible that locks can be avoided on the partitions.

## Lock promotion

***Definition:*** *Lock promotion* is the action of exchanging one lock on a resource for a more restrictive lock on the same resource, held by the same application process.

***Example:*** An application reads data, which requires an IS lock on a table space. Based on further calculation, the application updates the same data, which requires an IX lock on the table space. The application is said to *promote* the table space lock from mode IS to mode IX.

***Effects:*** When promoting the lock, DB2 first waits until any incompatible locks held by other processes are released. When locks are promoted, it is in the direction of increasing control over resources: from IS to IX, S, or X; from IX to SIX or X; from S to X; from U to X; and from SIX to X.

## Lock escalation

***Definition:*** *Lock escalation* is the act of releasing a large number of page, row or LOB locks, held by an application process on a single table or table space, to acquire a table or table space lock, or a set of partition locks, of mode S or X instead. When it occurs, DB2 issues message DSNI031I, which identifies the table space for which lock escalation occurred, and some information to help you identify what plan or package was running when the escalation occurred.

Lock counts are always kept on a table or table space level. For an application process that is accessing LOBs, the LOB lock count on the LOB table space is maintained separately from the base table space, and lock escalation occurs separately from the base table space.

When escalation occurs for a table space defined with LOCKPART YES, only partitions that are currently locked are escalated. Unlocked partitions remain unlocked. After lock escalation occurs, any unlocked partitions that are subsequently accessed are locked with a gross lock.

For an application process that is using Sysplex query parallelism, the lock count is maintained on a member basis, not globally across the group for the process. Thus, escalation on a table space or table by one member does not cause escalation on other members.

***Example:*** Assume that a segmented table space is defined with LOCKSIZE ANY and LOCKMAX 2000. DB2 can use page locks for a process that accesses a table in the table space and can escalate those locks. If the process attempts to lock more than 2000 pages in the table at one time, DB2 promotes its intent locks on the table to mode S or X and then releases its page locks.

If the process is using Sysplex query parallelism and a table space that it accesses has a LOCKMAX value of 2000, lock escalation occurs for a member only if more than 2000 locks are acquired for that member.

***When it occurs:*** Lock escalation balances concurrency with performance by using page or row locks while a process accesses relatively few pages or rows, and then changing to table space, table, or partition locks when the process accesses many. When it occurs, lock escalation varies by table space, depending on the values of LOCKSIZE and LOCKMAX, as described in

- "LOCKSIZE clause of CREATE and ALTER TABLESPACE" on page 671
- "LOCKMAX clause of CREATE and ALTER TABLESPACE" on page 672

Lock escalation is suspended during the execution of SQL statements for ALTER, CREATE, DROP, GRANT, and REVOKE.

See "Controlling LOB lock escalation" on page 695 for information about lock escalation for LOBs.

***Recommendations:*** The DB2 statistics and performance traces can tell you how often lock escalation has occurred and whether it has caused timeouts or deadlocks. As a rough estimate, if one quarter of your lock escalations cause timeouts or deadlocks, then escalation is not effective for you. You might alter the table to increase LOCKMAX and thus decrease the number of escalations.

Alternatively, if lock escalation is a problem, use LOCKMAX 0 to disable lock escalation. However, acquiring too many locks can cause DB2 to fail if IRLM runs out of storage for the locks. If you use LOCKSIZE ANY LOCKMAX 0 to disable lock escalation, DB2 might acquire an X lock on the table space instead of any page or row locks. To avoid the table space lock in these cases, alter the table space to increase LOCKMAX to a large value.

***Example:*** Assume that a table space is used by transactions that require high concurrency and that a batch job updates almost every page in the table space. For high concurrency, you should probably create the table space with LOCKSIZE PAGE and make the batch job commit every few seconds.

LOCKSIZE ANY is a possible choice, if you take other steps to avoid lock escalation. If you use LOCKSIZE ANY, specify a LOCKMAX value large enough so that locks held by transactions are not normally escalated. Also, LOCKS PER USER must be large enough so that transactions do not reach that limit.

If the batch job is:

- *Concurrent* with transactions, then it must use page or row locks and commit frequently: for example, every 100 updates. Review LOCKS PER USER to avoid exceeding the limit. The page or row locking uses significant processing time. Binding with ISOLATION(CS) may discourage lock escalation to an X table space lock for those applications that read a lot and update occasionally. However, this may not prevent lock escalation for those applications that are update intensive.

- *Non-concurrent* with transactions, then it need not use page or row locks. The application could explicitly lock the table in exclusive mode, described under "The statement LOCK TABLE" on page 690.

### Modes of transaction locks for various processes

The rows in Table 93 show a sample of several types of DB2 processes. The columns show the most restrictive mode of locks used for different objects and the possible conflicts between application processes.

*Table 93. Modes of DB2 transaction locks*

| Process | Catalog table spaces | Skeleton tables (SKCT and SKPT) | Database descriptor (DBD) (1) | Target table space (2) |
|---|---|---|---|---|
| Transaction with static SQL | IS (3) | S | n/a (4) | Any (5) |
| Query with dynamic SQL | IS (6) | S | S | Any (5) |
| BIND process | IX | X | S | n/a |
| SQL CREATE TABLE statement | IX | n/a | X | n/a |
| SQL ALTER TABLE statement | IX | X (7) | X | n/a |
| SQL ALTER TABLESPACE statement | IX | X (9) | X | n/a |
| SQL DROP TABLESPACE statement | IX | X (8) | X | n/a |
| SQL GRANT statement | IX | n/a | n/a | n/a |
| SQL REVOKE statement | IX | X (8) | n/a | n/a |

***Notes for Table 93:***

1. In a lock trace, these locks usually appear as locks on the DBD.
2. The target table space is one of the following table spaces:
   - Accessed and locked by an application process
   - Processed by a utility
   - Designated in the data definition statement
3. The lock is held briefly to check EXECUTE authority.
4. If the required DBD is not already in the EDM pool, locks are acquired on table space DBD01, which effectively locks the DBD.
5. For details, see Table 92 on page 660.
6. Except while checking EXECUTE authority, IS locks on catalog tables are held until a commit point.
7. The plan or package using the SKCT or SKPT is marked invalid if a referential constraint (such as a new primary key or foreign key) is added or changed, or the AUDIT attribute is added or changed for a table.
8. The plan or package using the SKCT or SKPT is marked invalid as a result of this operation.
9. These locks are not held when ALTER TABLESPACE is changing the following options: PRIQTY, SECQTY, PCTFREE, FREEPAGE, CLOSE, and ERASE.

## Lock tuning

This section describes what you can change to affect transaction locks, under:

- "Startup procedure options" on page 665
- "Installation options for wait times" on page 665
- "Other options that affect locking" on page 670
- "Bind options" on page 675

- “Isolation overriding with SQL statements” on page 689
- “The statement LOCK TABLE” on page 690

## Startup procedure options

The values of these options are passed to the startup procedure for the DB2 internal resource lock manager (IRLM) when you issue the MVS command START *irlmproc*.

### Using options for DB2 locking

The options relevant to DB2 locking are:

**SCOPE** Whether IRLM is used for data sharing (GLOBAL) or not (LOCAL). Use LOCAL unless you are using data sharing. If you use data sharing, specify GLOBAL.

**DEADLOK** The two values of this option specify:

1. The number of seconds between two successive scans for a local deadlock
2. The number of local scans that occur before a scan for global deadlock starts

**PC** Whether the IRLM locks are in the IRLM private address space (YES) or in the extended common storage area (NO). If YES, DB2 uses cross memory for IRLM requests.

**MAXCSA** The maximum amount of storage in the ECSA and CSA that IRLM uses for locks. In a display of the IRLM storage, this storage is called “accountable” storage, because it is accountable against the value you set for MAXCSA.

ECSA is a shared area of which DB2 is not the only user; it is used until no space is left, and then CSA is used. Lock requests that need more storage are rejected and the corresponding unit of work is rolled back.

### Estimating the storage needed for locks

For a conservative figure, assume:

- 250 bytes of storage for each lock.
- All concurrent threads hold the maximum number of row or page locks (LOCKS PER USER on installation panel DSNTIPJ). The number of table and table space locks is negligible.
- The maximum number of concurrent threads are active.

Then calculate: `Storage = 250 × (LOCKS PER USER) × (MAX USERS).`

That value is calculated when DB2 is installed. A warning message is issued if the value of MAXCSA is less than the calculated value. That result might mean rejecting lock requests.

## Installation options for wait times

These options determine how long it takes DB2 to identify that a process must be timed out or is deadlocked. They affect locking in your entire DB2 subsystem.

The following fields of the installation panels are relevant to transaction locks:

- “DEADLOCK TIME on installation panel DSNTIPJ” on page 666
- “RESOURCE TIMEOUT on installation panel DSNTIPI” on page 666
- “IDLE THREAD TIMEOUT on installation panel DSNTIPR” on page 668

The following field is relevant to drain locks:

- "UTILITY TIMEOUT on installation panel DSNTIPI" on page 668

## DEADLOCK TIME on installation panel DSNTIPJ

***Effect:*** DB2 scans for deadlocked processes at regular intervals. This field sets the length of the interval, in seconds.

***Default:*** 5 seconds.

***Recommendation:*** Determining the best value for deadlock detection is dependent on your workload. Deadlock detection can cause latch suspensions; therefore, for systems in which deadlocking is not a problem, have deadlock detection run less frequently for the best performance and concurrency (but don't choose a number greater than 5). However, if your system is prone to deadlocks, you want those detected as quickly as possible. In that case, choose 1.

## RESOURCE TIMEOUT on installation panel DSNTIPI

***Effect:*** Specifies a minimum number of seconds before a timeout can occur. A small value can cause a large number of timeouts. With a larger value, suspended processes more often resume normally, but they remain inactive for longer periods.

***Default:*** 60 seconds.

***Recommendation:*** If you can allow a suspended process to remain inactive for 60 seconds, use the defaults for both RESOURCE TIMEOUT and DEADLOCK TIME. To specify a different inactive period, you must consider how DB2 times out a process that is waiting for a transaction lock, as described in "Wait time for transaction locks", below.

### Wait time for transaction locks

In a scanning schedule to determine whether a process waiting for a transaction lock has timed out, DB2 uses the following two factors:

- A timeout period
- An operation multiplier

***The timeout period:*** From the value of RESOURCE TIMEOUT and DEADLOCK TIME, DB2 calculates a timeout period as shown below. Assume that DEADLOCK TIME is 5 and RESOURCE TIMEOUT is 18.

1. Divide RESOURCE TIMEOUT by DEADLOCK TIME (18/5 = 3.6). IRLM limits the result of this division to 255.
2. Round the result to the next largest integer (Round up 3.6 to 4).
3. Multiply the DEADLOCK TIME by that integer (4 × 5 = 20).

The result, the timeout period (20 seconds), is always at least as large as the value of RESOURCE TIMEOUT (18 seconds), except when the RESOURCE TIMEOUT divided by DEADLOCK TIME exceeds 255.

***The timeout multiplier:*** Requests from different types of processes wait for different multiples of the timeout period. In a data sharing environment, you can add another multiplier to those processes to wait for retained locks.

In some cases, you can modify the multiplier value. Table 94 on page 667 indicates the multiplier by type of process, and whether you can change it.

*Table 94. Timeout multiplier by type*

| Type | Multiplier | Modifiable? |
|---|---|---|
| IMS MPP, IMS Fast Path Message Processing, CICS, QMF, CAF, TSO batch and online | 1 | No |
| IMS BMPs | 4 | Yes |
| IMS DL/I batch | 6 | Yes |
| IMS Fast Path Non-message processing | 6 | No |
| BIND subcommand processing | 3 | No |
| STOP DATABASE command processing | 10 | No |
| Utilities | 6 | Yes |
| Retained locks for all types | 0 | Yes |

See "UTILITY TIMEOUT on installation panel DSNTIPI" on page 668 for information about modifying the utility timeout multiplier. See "Additional multiplier for retained locks" for information about creating an additional multiplier for retained lock timeout.

***Changing the multiplier for IMS BMP and DL/I batch:*** You can modify the multipliers for IMS BMP and DL/I batch by modifying the following subsystem parameters on installation panel DSNTIPI:

**IMS BMP TIMEOUT** The timeout multiplier for IMS BMP connections. A value from 1-254 is acceptable. The default is 4.

**DL/I BATCH TIMEOUT** The timeout multiplier for IMS DL/I batch connections. A value from 1-254 is acceptable. The default is 6.

***Additional multiplier for retained locks:*** For data sharing, you can specify an additional timeout multiplier to be applied to the connection's normal timeout multiplier. This multiplier is used when the connection is waiting for a retained lock, which is a lock held by a failed member of a data sharing group. A zero means don't wait for retained locks. Chapter 2 of *DB2 Data Sharing: Planning and Administration* for more information about retained locks.

***The scanning schedule:*** Figure 77 on page 668 illustrates the following example of scanning to detect a timeout:

- DEADLOCK TIME has the default value of 5 seconds.
- RESOURCE TIMEOUT was chosen to be 18 seconds. Hence, the timeout period is 20 seconds, as described above.
- A bind operation starts 4 seconds before the next scan. The operation multiplier for a bind operation is 3.

The scans proceed through the following steps:

1. A scan starts 4 seconds after the bind operation requests a lock. As determined by the DEADLOCK TIME, scans occur every 5 seconds. The first scan in the example detects that the operation is inactive.
2. IRLM allows at least one full interval of DEADLOCK TIME as a "grace period" for an inactive process. After that, its lock request is judged to be waiting. At 9 seconds, the second scan detects that the bind operation is waiting.

3. The bind operation continues to wait for a multiple of the timeout period. In the example, the multiplier is 3 and the timeout period is 20 seconds. The bind operation continues to wait for 60 seconds longer.
4. The scan that starts 69 seconds after the bind operation detects that the process has timed out.

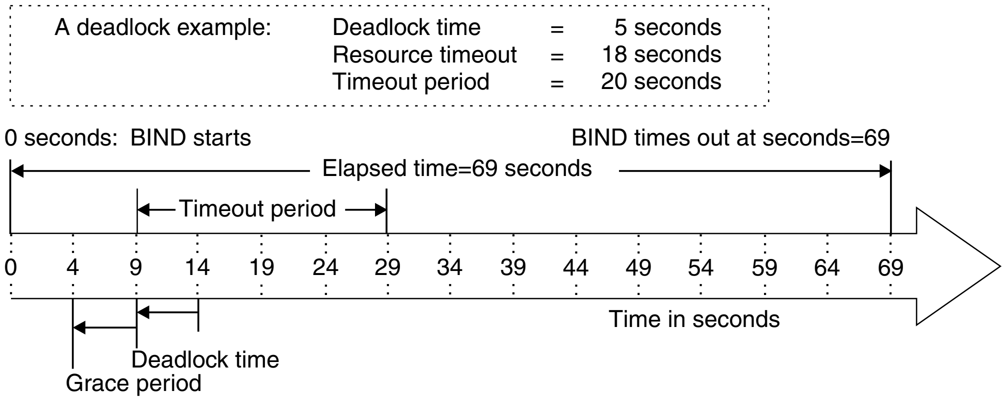


*Figure 77. An example of scanning for timeout*

***Effect:*** An operation can remain inactive for longer than the value of RESOURCE TIMEOUT.

If you are in a data sharing environment, the deadlock and timeout detection process is longer than that for non-data-sharing systems. See Chapter 6 of *DB2 Data Sharing: Planning and Administration* for more information about global detection processing and elongation of the timeout period.

***Recommendation:*** Consider the length of inaction time when choosing your own values of DEADLOCK TIME and RESOURCE TIMEOUT.

## IDLE THREAD TIMEOUT on installation panel DSNTIPR

***Effect:*** Specifies a period for which an active distributed thread can hold locks without doing any processing. After that period, a regular scan (at 3-minute intervals) detects that the thread has been idle for the specified period, and DB2 cancels the thread.

The cancellation applies only to active threads. If your installation permits distributed threads to be inactive and hold no resources, those threads are allowed to remain idle indefinitely.

***Default:*** 0. That value disables the scan to time out idle threads. The threads can then remain idle indefinitely.

***Recommendation:*** If you have experienced distributed users leaving an application idle while it holds locks, pick an appropriate value other than 0 for this period. Because the scan occurs only at 3-minute intervals, your idle threads will generally remain idle for somewhat longer than the value you specify.

## UTILITY TIMEOUT on installation panel DSNTIPI

***Effect:*** Specifies an operation multiplier for utilities waiting for a drain lock, for a transaction lock, or for claims to be released.

***Default:*** 6.

***Recommendation:*** With the default value, a utility generally waits longer for a resource than does an SQL application. To specify a different inactive period, you must consider how DB2 times out a process that is waiting for a drain, as described in “Wait time for drains”.

## Wait time for drains

A process that requests a drain might wait for two events:

1. Acquiring the drain lock. If another user holds the needed drain lock in an incompatible lock mode, then the drainer waits.
2. Releasing all claims on the object. Even after the drain lock is acquired, the drainer waits until all claims are released before beginning to process.

If the process drains more than one claim class, it must wait for those events to occur for *each claim class* it drains.

Hence, to calculate the maximum amount of wait time:

1. Start with the wait time for a drain lock.
2. Add the wait time for claim release.
3. Multiply the result by the number of claim classes drained.

***Wait times for drain lock and claim release:*** Both wait times are based on the timeout period that is calculated in “RESOURCE TIMEOUT on installation panel DSNTIPI” on page 666. For the REORG utility with the SHRLEVEL REFERENCE or SHRLEVEL CHANGE option, you can use utility parameters to specify the wait time for a drain lock and to indicate if additional attempts should be made to acquire the drain lock. For more information, see *DB2 Utility Guide and Reference*.

| **Drainer** | **Each wait time is:** |
|---|---|
| **Utility** | (timeout period) × (value of UTILITY TIMEOUT) |
| **Other process** | timeout period |

***Maximum wait time:*** Because the maximum wait time for a drain lock is the same as the maximum wait time for releasing claims, you can calculate the total maximum wait time as follows:

For **utilities**:

```
2 × (timeout period) × (UTILITY TIMEOUT) × (number of claim classes)
```

For **other processes**:

```
2 × (timeout period) × (operation multiplier) × (number of claim classes)
```

***Example:*** How long might the LOAD utility be suspended before being timed out? LOAD must drain 3 claim classes. If:

Timeout period = 20
Value of UTILITY TIMEOUT = 6

Then:

Maximum wait time = 2 × 20 × 6 × 3

or:

Maximum wait time = 720 seconds

***Wait times less than maximum:*** The maximum drain wait time is the longest possible time a drainer can wait for a drain, *not* the length of time it always waits.

***Example:*** Table 95 lists the steps LOAD takes to drain the table space and the maximum amount of wait time for each step. A timeout can occur at any step. At step 1, the utility can wait 120 seconds for the repeatable read drain lock. If that lock is not available by then, the utility times out after 120 seconds. It does not wait 720 seconds.

*Table 95. Maximum drain wait times: LOAD utility*

| Step | Maximum Wait Time (seconds) |
|---|---|
| 1. Get repeatable read drain lock | 120 |
| 2. Wait for all RR claims to be released | 120 |
| 3. Get cursor stability read drain lock | 120 |
| 4. Wait for all CS claims to be released | 120 |
| 5. Get write drain lock | 120 |
| 6. Wait for all write claims to be released | 120 |
| **Total** | **720** |

## Other options that affect locking

You can use various options to control such things as how many locks are used and which mode is used for certain locks. This section describes the following installation options and SQL statement clauses:

- “LOCKS PER USER field of installation panel DSNTIPJ”
- “LOCKSIZE clause of CREATE and ALTER TABLESPACE” on page 671
- “LOCKMAX clause of CREATE and ALTER TABLESPACE” on page 672
- “LOCKS PER TABLE(SPACE) field of installation panel DSNTIPJ” on page 673
- “The option U LOCK FOR RR/RS” on page 673
- “Option to release locks for cursors defined WITH HOLD” on page 673
- “Option XLOCK for searched updates/deletes” on page 674
- “Option to avoid locks during predicate evaluation” on page 674

See “Effects of table spaces of different types” on page 651 for information about the LOCKPART clause of CREATE and ALTER TABLESPACE.

### LOCKS PER USER field of installation panel DSNTIPJ

***Effect:*** Specifies the maximum number of page, row, or LOB locks that can be held by a single process at any one time. It includes locks for both the DB2 catalog and directory and for user data.

When a request for a page, row, or LOB lock exceeds the specified limit, it receives SQLCODE -904: “resource unavailable” (SQLSTATE '57011'). The requested lock cannot be acquired until some of the existing locks are released.

***Default:*** 10000

***Recommendation:*** The default should be adequate for 90 percent of the work load when using page locks. If you use row locks on very large tables, you might want a higher value. If you use LOBs, you might need a higher value.

Review application processes that require higher values to see if they can use table space locks rather than page, row, or LOB locks. The accounting trace shows the maximum number of page, row, or LOB locks a process held while running.

## LOCKSIZE clause of CREATE and ALTER TABLESPACE

The information under this heading, up to “LOCKMAX clause of CREATE and ALTER TABLESPACE” on page 672, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

***Effect:*** Specifies the size for locks held on a table or table space by any application process that accesses it. The options are:

**LOCKSIZE TABLESPACE**
A process acquires no table, page, row, or LOB locks within the table space. That improves performance by reducing the number of locks maintained, but greatly inhibits concurrency.

**LOCKSIZE TABLE**
A process acquires table locks on tables in a segmented table space. If the table space contains more than one table, this option can provide acceptable concurrency with little extra cost in processor resources.

**LOCKSIZE PAGE**
A process acquires page locks, plus table, partition, or table space locks of modes that permit page locks (IS, IX, or SIX). The effect is not absolute: a process can still acquire a table, partition, or table space lock of mode S or X, without page locks, if that is needed. In that case, the bind process issues a message warning that the lock size has been promoted as described under “Lock promotion” on page 662.

**LOCKSIZE ROW**
A process acquires row locks, plus table, partition, or table space locks of modes that permit row locks (IS, IX, or SIX). The effect is not absolute: a process can still acquire a table or table space lock of mode S or X, without row locks, if that is needed. In that case, the bind process issues a message warning that the lock size has been promoted as described under “Lock promotion” on page 662.

**LOCKSIZE ANY**
DB2 chooses the size of the lock, usually LOCKSIZE PAGE.

**LOCKSIZE LOB**
If a LOB must be accessed, a process acquires LOB locks and the necessary LOB table space locks (IS or IX). This option is valid only for LOB table spaces. See “LOB locks” on page 691 for more information about LOB locking.

DB2 attempts to acquire an S lock on table spaces that are started with read-only access. If the LOCKSIZE is PAGE or ROW, and DB2 cannot get the S lock, it requests an IS lock. If a partition is started with read-only access, and the table space is defined with LOCKPART YES, then DB2 attempts to get an S lock on the partition that is started RO. For a complete description of how the LOCKSIZE clause affects lock attributes, see “DB2’s choice of lock types” on page 659.

***Default:*** LOCKSIZE ANY

***Catalog record:*** Column LOCKRULE of table SYSIBM.SYSTABLESPACE.

***Recommendation:*** If you do not use the default, base your choice upon the results of monitoring applications that use the table space.

**Row locks or page locks?** The question of whether to use row or page locks depends on your data and your applications. If you are experiencing contention on data pages of a table space now defined with LOCKSIZE PAGE, consider LOCKSIZE ROW. But consider also the trade-offs.

The resource required to acquire, maintain, and release a row lock is about the same as that required for a page lock. If your data has 10 rows per page, a table space scan or an index scan can require nearly 10 times as much resource for row locks as for page locks. But locking only a row at a time, rather than a page, might reduce the chance of contention with some other process by 90%, especially if access is random. (Row locking is not recommended for sequential processing.)

In many cases, DB2 can avoid acquiring a lock when reading data that is known to be committed. Thus, if only 2 of 10 rows on a page contain uncommitted data, DB2 must lock the entire page when using page locks, but might ask for locks on only the 2 rows when using row locks. Then, the resource required for row locks would be only twice as much, not 10 times as much, as that required for page locks.

On the other hand, if two applications update the same rows of a page, and not in the same sequence, then row locking might even increase contention. With page locks, the second application to access the page must wait for the first to finish and might time out. With row locks, the two applications can access the same page simultaneously, and might deadlock while trying to access the same set of rows.

In short, no single answer fits all cases.

## LOCKMAX clause of CREATE and ALTER TABLESPACE

The information under this heading, up to “LOCKS PER TABLE(SPACE) field of installation panel DSNTIPJ” on page 673, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

***Effects of the options:*** You can specify these values not only for tables of user data but also, by using ALTER TABLESPACE, for tables in the DB2 catalog.

**LOCKMAX** ***n***

Specifies the maximum number of page or row locks that a single application process can hold on the table space before those locks are escalated as described in “Lock escalation” on page 662. For LOB table spaces, this value specifies the number of LOB locks that the application process can hold before escalating. For an application that uses Sysplex query parallelism, a lock count is maintained on each member.

**LOCKMAX SYSTEM**

Specifies that *n* is effectively equal to the system default set by the field LOCKS PER TABLE(SPACE) of installation panel DSNTIPJ.

**LOCKMAX 0**

Disables lock escalation entirely.

***Default:*** Depends on the value of LOCKSIZE, as follows:

| LOCKSIZE | Default for LOCKMAX |
|---|---|
| ANY | SYSTEM |
| other | 0 |

***Catalog record:*** Column LOCKMAX of table SYSIBM.SYSTABLESPACE.

***Recommendations:*** If you do not use the default, base your choice upon the results of monitoring applications that use the table space.

Aim to set the value of LOCKMAX high enough that, when lock escalation occurs, one application already holds so many locks that it significantly interferes with others. For example, if an application holds half a million locks on a table with a million rows, it probably already locks out most other applications. Yet lock escalation can prevent it from potentially acquiring another half million locks.

If you alter a table space from LOCKSIZE PAGE or LOCKSIZE ANY to LOCKSIZE ROW, consider increasing LOCKMAX to allow for the increased number of locks that applications might require.

| If you use LOCKSIZE ANY LOCKMAX 0 to disable lock escalation, DB2 might acquire an X lock on the table space instead of any page or row locks. To avoid the table space lock in these cases, alter the table space to increase LOCKMAX to a large value.

## LOCKS PER TABLE(SPACE) field of installation panel DSNTIPJ

***Effect:*** The value becomes the default value (SYSTEM) for the LOCKMAX clause of the SQL statements CREATE TABLESPACE and ALTER TABLESPACE.

***Default:*** 1000

***Recommendation:*** Use the default or, if you are migrating from a previous release of DB2, continue to use the existing value. When you create or alter a table space, especially when you alter one to use row locks, use the LOCKMAX clause explicitly for that table space.

## The option U LOCK FOR RR/RS

This option, on installation panel DSNTIPI, determines the mode of the lock first acquired on a row or page, table, partition, or table space for certain statements bound with RR or RS isolation. Those statements include:

- SELECT with FOR UPDATE OF

  If you specify the SELECT using WITH RS KEEP UPDATE LOCKS or WITH RR KEEP UPDATE LOCKS, an X lock is held on the rows or pages.
- UPDATE and DELETE, without a cursor

| Option Value | Lock Mode |
|---|---|
| NO (default) | S |
| YES | U or X |

YES can avoid deadlocks but reduces concurrency.

## Option to release locks for cursors defined WITH HOLD

***Effect:*** The RELEASE LOCKS field of installation panel DSNTIP4 lets you indicate that you want DB2 to release a data page or row lock after a COMMIT is issued for cursors defined WITH HOLD. This lock is not necessary for maintaining cursor position.

***Default:*** YES

***Recommendation:*** The default, YES, causes DB2, at commit time, to release the data page or row lock for the row on which the cursor is positioned. This lock is unnecessary for maintaining cursor position. To improve concurrency, specify YES. Specify NO only for those cases in which existing applications rely on that particular data lock.

## Option XLOCK for searched updates/deletes

***Effect:*** A subsystem parameter XLKUPDLT lets you disable update lock (ULOCK) on searched UPDATEs and DELETEs so that you do not have to issue a second lock request to upgrade the lock from U to X (exclusive lock) for each updated row.

***Default:*** NO

***Recommendation:*** This feature is primarily beneficial in a data sharing environment. It should be used when most or all searched UPDATEs/DELETEs use an index or can be evaluated by stage 1 processing. When NO is specified, DB2 uses an S lock or a U lock when scanning for qualifying rows. The lock on any qualifying row or page is then upgraded to an X lock before performing the update or delete. For non-qualifying rows or pages, the lock is released if ISOLATION(CS) is used. For ISOLATION(RS) or ISOLATION(RR), an S lock is retained on the row or page until the next commit point. This option is best for achieving the highest rates of concurrency.

When YES is specified, DB2 uses an X lock on rows or pages that qualify during stage 1 processing. With ISOLATION(CS), the lock is released if the row or page is not updated or deleted because it is rejected by stage 2 processing. With ISOLATION(RR) or ISOLATION(RS), DB2 acquires an X lock on all rows that fall within the range of the selection expression. Thus, a lock upgrade request is not needed for qualifying rows though the lock duration is changed from manual to commit. The lock duration change is not as costly as a lock upgrade.

## Option to avoid locks during predicate evaluation

***Effect:*** The EVALUATE UNCOMMITTED field of installation panel DSNTIP4 indicates if predicate evaluation can occur on uncommitted data of other transactions. The option applies only to stage 1 predicate processing that uses table access (table space scan, index-to-data access, and RID list processing) for queries with isolation level RS or CS.

Although this option influences whether predicate evaluation can occur on uncommitted data, it does not influence whether uncommitted data is returned to an application. Queries with isolation level RS or CS will return only committed data. They will never return the uncommitted data of other transactions, even if predicate evaluation occurs on such. If data satisfies the predicate during evaluation, the data is locked as needed, and the predicate is re-evaluated as needed before the data is returned to the application.

A value of NO specifies that predicate evaluation occurs only on committed data (or on the application's own uncommitted changes). NO ensures that all qualifying data is always included in the answer set.

A value of YES specifies that predicate evaluation can occur on uncommitted data of other transactions. With YES, data might be excluded from the answer set. Data that does not satisfy the predicate during evaluation but then, because of undo processing (ROLLBACK or statement failure), reverts to a state that does satisfy the predicate is missing from the answer set. A value of YES enables DB2 to take fewer locks during query processing. The number of locks avoided depends on:
- The query's access path
- The number of evaluated rows that do not satisfy the predicate
- The number of those rows that are on overflow pages

***Default:*** NO

***Recommendation:*** Specify YES to improve concurrency if your applications can tolerate returned data to falsely exclude any data that would be included as the result of undo processing (ROLLBACK or statement failure).

# Bind options

The information under this heading, up to “Isolation overriding with SQL statements” on page 689, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

These options determine when an application process acquires and releases its locks and to what extent it isolates its actions from possible effects of other processes acting concurrently.

These options of bind operations are relevant to transaction locks:
- “The ACQUIRE and RELEASE options”
- “The ISOLATION option” on page 678
- “The CURRENTDATA option” on page 685

## The ACQUIRE and RELEASE options

***Effects:*** The ACQUIRE and RELEASE options of bind determine when DB2 locks an object (table, partition, or table space) your application uses and when it releases the lock. (The ACQUIRE and RELEASE options do not affect page, row, or LOB locks.) The options apply to static SQL statements, which are bound before your program executes. If your program executes dynamic SQL statements, the objects they lock are locked when first accessed and released at the next commit point though some locks acquired for dynamic SQL may be held past commit points. See “The RELEASE option and dynamic statement caching” on page 676.

| Option | Effect |
|---|---|
| **ACQUIRE(ALLOCATE)** | Acquires the lock when the object is allocated. This option is not allowed for BIND or REBIND PACKAGE. |
| **ACQUIRE(USE)** | Acquires the lock when the object is first accessed. |
| **RELEASE(DEALLOCATE)** | Releases the lock when the object is deallocated (the application ends). The value has no effect on dynamic SQL statements, which always use RELEASE(COMMIT), unless you are using dynamic statement caching. For information about the RELEASE option with dynamic statement caching, see “The RELEASE option and dynamic statement caching” on page 676. |
| **RELEASE(COMMIT)** | Releases the lock at the next commit point, unless there are held cursors or held locators. If the application accesses the object again, it must acquire the lock again. |

***Example:*** An application selects employee names and telephone numbers from a table, according to different criteria. Employees can update their own telephone numbers. They can perform several searches in succession. The application is bound with the options ACQUIRE(USE) and RELEASE(DEALLOCATE), for these reasons:

- The alternative to ACQUIRE(USE), ACQUIRE(ALLOCATE), gets a lock of mode IX on the table space as soon as the application starts, because that is needed if an update occurs. But most uses of the application do not update the table and so need only the less restrictive IS lock. ACQUIRE(USE) gets the IS lock when the table is first accessed, and DB2 promotes the lock to mode IX if that is needed later.

- Most uses of this application do not update and do not commit. For those uses, there is little difference between RELEASE(COMMIT) and RELEASE(DEALLOCATE). But administrators might update several phone numbers in one session with the application, and the application commits after each update. In that case, RELEASE(COMMIT) releases a lock that DB2 must acquire again immediately. RELEASE(DEALLOCATE) holds the lock until the application ends, avoiding the processing needed to release and acquire the lock several times.

***Effect of LOCKPART YES:*** Partition locks follow the same rules as table space locks, and *all* partitions are held for the same duration. Thus, if one package is using RELEASE(COMMIT) and another is using RELEASE(DEALLOCATE), all partitions use RELEASE(DEALLOCATE).

***The RELEASE option and dynamic statement caching:*** Generally, the RELEASE option has no effect on dynamic SQL statements with one exception. When you use the bind options RELEASE(DEALLOCATE) and KEEPDYNAMIC(YES), and your subsystem is installed with YES for field CACHE DYNAMIC SQL on panel DSNTIP4, DB2 retains prepared SELECT, INSERT, UPDATE, and DELETE statements in memory past commit points. For this reason, DB2 can honor the RELEASE(DEALLOCATE) option for these dynamic statements. The locks are held until deallocation, or until the commit after the prepared statement is freed from memory, in the following situations:

- The application issues a PREPARE statement with the same statement identifier.
- The statement is removed from memory because it has not been used.
- An object that the statement is dependent on is dropped or altered, or a privilege needed by the statement is revoked.
- RUNSTATS is run against an object that the statement is dependent on.

If a lock is to be held past commit and it is an S, SIX, or X lock on a table space or a table in a segmented table space, DB2 sometimes demotes that lock to an intent lock (IX or IS) at commit. DB2 demotes a gross lock if it was acquired for one of the following reasons:

- DB2 acquired the gross lock because of lock escalation.
- The application issued a LOCK TABLE.
- The application issued a mass delete (DELETE FROM ... without a WHERE clause).

For table spaces defined as LOCKPART YES, lock demotion occurs as with other table spaces; that is, the lock is demoted at the table space level, not the partition level.

***Defaults:*** The defaults differ for different types of bind operations:

| **Operation** | **Default values** |
|---|---|
| **BIND PLAN** | ACQUIRE(USE) and RELEASE(COMMIT). |
| **BIND PACKAGE** | There is no option for ACQUIRE; ACQUIRE(USE) is always used. At the local server the default for RELEASE is the value used by the plan that |

includes the package in its package list. At a remote server the default is COMMIT.

**REBIND PLAN or PACKAGE** The existing values for the plan or package being rebound.

***Recommendation:*** Choose a combination of values for ACQUIRE and RELEASE based on the characteristics of the particular application.

***The RELEASE option and DDL operations for remote requesters:*** When you perform DDL operations on behalf of remote requesters and RELEASE(DEALLOCATE) is in effect, be aware of the following condition. When a package that is bound with RELEASE(DEALLOCATE) accesses data at a server, it might prevent other remote requesters from performing CREATE, ALTER, DROP, GRANT, or REVOKE operations at the server.

To allow those operations to complete, you can use the command STOP DDF MODE(SUSPEND). The command suspends server threads and terminates their locks so that DDL operations from remote requesters can complete. When these operations complete, you can use the command START DDF to resume the suspended server threads. However, even after the command STOP DDF MODE(SUSPEND) completes successfully, database resources might be held if DB2 is performing any activity other than inbound DB2 processing. You might have to use the command CANCEL THREAD to terminate other processing and thereby free the database resources.

## Advantages and disadvantages of the combinations

***ACQUIRE(ALLOCATE) / RELEASE(DEALLOCATE):*** In some cases, this combination can avoid deadlocks by locking all needed resources as soon as the program starts to run. This combination is most useful for a long-running application that runs for hours and accesses various tables, because it prevents an untimely deadlock from wasting that processing.

- All tables or table spaces used in DBRMs bound directly to the plan are locked when the plan is allocated.
- All tables or table spaces are unlocked only when the plan terminates.
- The locks used are the most restrictive needed to execute all SQL statements in the plan regardless of whether the statements are actually executed.
- Restrictive states are not checked until the page set is accessed. Locking when the plan is allocated insures that the job is compatible with other SQL jobs. Waiting until the first access to check restrictive states provides greater availability; however, it is possible that an SQL transaction could:
  - Hold a lock on a table space or partition that is stopped
  - Acquire a lock on a table space or partition that is started for DB2 utility access only (ACCESS(UT))
  - Acquire an exclusive lock (IX, X) on a table space or partition that is started for read access only (ACCESS(RO)), thus prohibiting access by readers

***Disadvantages:*** This combination reduces concurrency. It can lock resources in high demand for longer than needed. Also, the option ACQUIRE(ALLOCATE) turns off selective partition locking; if you are accessing a table space defined with LOCKPART YES, all partitions are locked.

***Restriction:*** This combination is not allowed for BIND PACKAGE. Use this combination if processing efficiency is more important than concurrency. It is a good choice for batch jobs that would release table and table space locks only to reacquire them almost immediately. It might even improve concurrency, by allowing batch jobs to finish sooner. Generally, do not use this combination if your application contains many SQL statements that are often not executed.

***ACQUIRE(USE) / RELEASE(DEALLOCATE):*** This combination results in the most efficient use of processing time in most cases.

- A table, partition, or table space used by the plan or package is locked only if it is needed while running.
- All tables or table spaces are unlocked only when the plan terminates.
- The least restrictive lock needed to execute each SQL statement is used, with the exception that if a more restrictive lock remains from a previous statement, that lock is used without change.

***Disadvantages:*** This combination can increase the frequency of deadlocks. Because all locks are acquired in a sequence that is predictable only in an actual run, more concurrent access delays might occur.

***ACQUIRE(USE) / RELEASE(COMMIT):*** This combination is the default combination and provides the greatest concurrency, but it requires more processing time if the application commits frequently.

- A table or table space is locked only when needed. That locking is important if the process contains many SQL statements that are rarely used or statements that are intended to access data only in certain circumstances.
- Table, partition, or table space locks are released at the next commit point unless the cursor is defined WITH HOLD. See “The effect of WITH HOLD for a cursor” on page 688 for more information.
- The least restrictive lock needed to execute each SQL statement is used except when a more restrictive lock remains from a previous statement. In that case, that lock is used without change.

***Disadvantages:*** This combination can increase the frequency of deadlocks. Because all locks are acquired in a sequence that is predictable only in an actual run, more concurrent access delays might occur.

***ACQUIRE(ALLOCATE) / RELEASE(COMMIT):*** This combination is not allowed; it results in an error message from BIND.

## The ISOLATION option

***Effects:*** Specifies the degree to which operations are isolated from the possible effects of other operations acting concurrently. Based on this information, DB2 releases S and U locks on rows or pages as soon as possible.

**Option Effect**

**ISOLATION(RR)**

*Repeatable read*: A row or page lock is held for all accessed rows, qualifying or not, at least until the next commit point. If the application process returns to the same page and reads the same row again, another application cannot have changed the rows nor have inserted any new qualifying rows.

The repeatability of the read is guaranteed only until the application commits. Even if a cursor is held on a specific row or page, the result set can change after a commit.

**ISOLATION(RS)**

*Read stability*: A row or page lock is held for pages or rows that are returned to an application at least until the next commit point. If a row or page is rejected during stage 2 processing, its lock is still held, even though it is not returned to the application.

If the application process returns to the same page and reads the same row again, another application cannot have changed the rows, although additional qualifying rows might have been inserted by another application process. A similar situation can also occur if a row or page that is not returned to the application is updated by another application process. If the row now satisfies the search condition, it appears.

When determining whether a row satisfies the search condition, DB2 can avoid taking the lock altogether if the row contains uncommitted data. If the row does not satisfy the predicate, lock avoidance is possible when the value of the EVALUATE UNCOMMITTED field of installation panel DSNTIP4 is YES. For details, see "Option to avoid locks during predicate evaluation" on page 674.

**ISOLATION(CS)**

*Cursor stability*: A row or page lock is held only long enough to allow the cursor to move to another row or page. For data that satisfies the search condition of the application, the lock is held until the application locks the next row or page. For data that does not satisfy the search condition, the lock is immediately released.

The data returned to an application that uses ISOLATION(CS) is committed, but if the application process returns to the same page, another application might have since updated or deleted the data, or might have inserted additional qualifying rows. This is especially true if DB2 returns data from a result table in a work file.

For example, if DB2 has to put an answer set in a result table (such as for a sort), DB2 releases the lock immediately after it puts the row or page in the result table in the work file. Using cursor stability, the base table can change while your application is processing the result of the sort output.

In some cases, DB2 can avoid taking the lock altogether, depending on the value of the CURRENTDATA bind option or the value of the EVALUATE UNCOMMITTED field on installation panel DSNTIP4.

- *Lock avoidance on committed data*: If DB2 can determine that the data it is reading has already been committed, it can avoid taking the lock altogether. For rows that do not satisfy the search condition, this lock avoidance is possible with CURRENTDATA(YES) or CURRENTDATA(NO). For rows that satisfy the search condition, lock avoidance is possible only when you use the option CURRENTDATA(NO). For more details, see "The CURRENTDATA option" on page 685.
- *Lock avoidance on uncommitted data*: For rows that do not satisfy the search condition, lock avoidance is possible when the value of EVALUATE UNCOMMITTED is YES. For details, see "Option to avoid locks during predicate evaluation" on page 674.

**ISOLATION(UR)**

*Uncommitted read*: The application acquires no page or row locks and can run concurrently with most other operations.[9] But the application is in danger of reading data that was changed by another operation but not yet committed. A UR application can acquire LOB locks, as described in “LOB locks” on page 691.

For restrictions on isolation UR, see “Restrictions” on page 684 for more information.

***Default:*** The default differs for different types of bind operations:

| **Operation** | **Default value** |
|---|---|
| **BIND PLAN** | ISOLATION(RR) |
| **BIND PACKAGE** | The value used by the plan that includes the package in its package list |
| **REBIND PLAN or PACKAGE** | The existing value for the plan or package being rebound |

For more detailed examples, see Part 4 of *DB2 Application Programming and SQL Guide*.

***Recommendations:*** Choose a value of ISOLATION based on the characteristics of the particular application.

## Advantages and disadvantages of the isolation values

The various isolation levels offer less or more concurrency at the cost of more or less protection from other application processes. The values you choose should be based primarily on the needs of the application. This section presents the isolation levels in order from the one offering the least concurrency (RR) to that offering the most (UR).

**ISOLATION (RR)**

Allows the application to read the same pages or rows more than once without allowing any UPDATE, INSERT, or DELETE by another process. All accessed rows or pages are locked, even if they do not satisfy the predicate.

Figure 78 on page 681 shows that all locks are held until the application commits. In the following example, the rows held by locks L2 and L4 satisfy the predicate.

[9] The exceptions are mass delete operations and utility jobs that drain all claim classes.

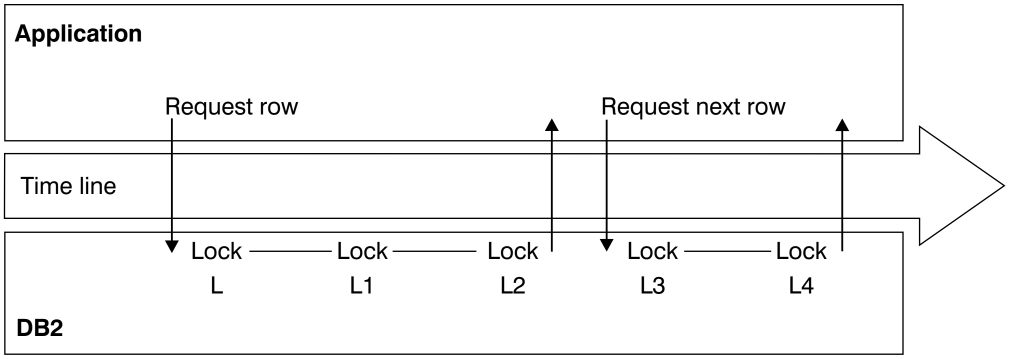


*Figure 78. How an application using RR isolation acquires locks. All locks are held until the application commits.*

Applications that use repeatable read can leave rows or pages locked for longer periods, especially in a distributed environment, and they can claim more logical partitions than similar applications using cursor stability.

Applications that use repeatable read and access a nonpartitioning index cannot run concurrently with utility operations that drain all claim classes of the nonpartitioning index, even if they are accessing different logical partitions. For example, an application bound with ISOLATION(RR) cannot update partition 1 while the LOAD utility loads data into partition 2. Concurrency is restricted because the utility needs to drain all the repeatable-read applications from the nonpartitioning index to protect the repeatability of the reads by the application.

Because so many locks can be taken, lock escalation might take place. Frequent commits release the locks and can help avoid lock escalation.

With repeatable read, lock promotion occurs for table space scan to prevent the insertion of rows that might qualify for the predicate. (If access is via index, DB2 locks the key range. If access is via table space scans, DB2 locks the table, partition, or table space.)

An installation option determines the mode of lock chosen for a cursor defined with the clause FOR UPDATE OF and bound with repeatable read. For details, see “The option U LOCK FOR RR/RS” on page 673.

**ISOLATION (RS)**

Allows the application to read the same pages or rows more than once without allowing qualifying rows to be updated or deleted by another process. It offers possibly greater concurrency than repeatable read, because although other applications cannot change rows that are returned to the original application, they can insert new rows or update rows that did not satisfy the original application’s search condition. Only those rows or pages that satisfy the stage 1 predicate (and all rows or pages evaluated during stage 2 processing) are locked until the application commits. Figure 79 on page 682 illustrates this. In the example, the rows held by locks L2 and L4 satisfy the predicate.

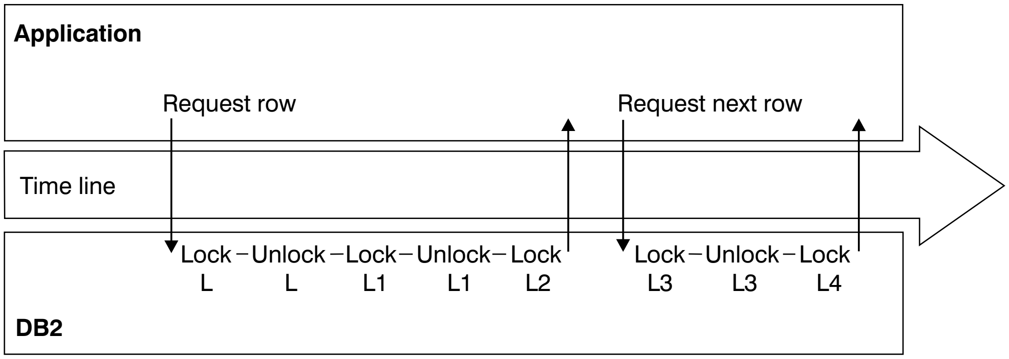


*Figure 79. How an application using RS isolation acquires locks when no lock avoidance techniques are used. Locks L2 and L4 are held until the application commits. The other locks aren't held.*

Applications using read stability can leave rows or pages locked for long periods, especially in a distributed environment.

If you do use read stability, plan for frequent commit points.

An installation option determines the mode of lock chosen for a cursor defined with the clause FOR UPDATE OF and bound with read stability. For details, see "The option U LOCK FOR RR/RS" on page 673.

**ISOLATION (CS)**

Allows maximum concurrency with data integrity. However, after the process leaves a row or page, another process can change the data. With CURRENTDATA(NO), the process doesn't have to leave a row or page to allow another process to change the data. If the first process returns to read the same row or page, the data is not necessarily the same. Consider these consequences of that possibility:

- For table spaces created with LOCKSIZE ROW, PAGE, or ANY, a change can occur even while executing a single SQL statement, if the statement reads the same row more than once. In the following example:

  ```
  SELECT * FROM T1
    WHERE COL1 = (SELECT MAX(COL1) FROM T1);
  ```

  data read by the inner SELECT can be changed by another transaction before it is read by the outer SELECT. Therefore, the information returned by this query might be from a row that is no longer the one with the maximum value for COL1.
- In another case, if your process reads a row and returns later to update it, that row might no longer exist or might not exist in the state that it did when your application process originally read it. That is, another application might have deleted or updated the row. **If your application is doing non-cursor operations on a row under the cursor, make sure the application can tolerate "not found" conditions**.

  Similarly, assume another application updates a row after you read it. If your process returns later to update it based on the value you originally read, you are, in effect, erasing the update made by the other process. **If you use isolation (CS) with update, your process might need to lock out concurrent updates.** One method is to declare a cursor with the clause FOR UPDATE OF.

**Product-sensitive Programming Interface**

For packages and plans that contain updatable scrollable cursors, ISOLATION(CS) lets DB2 use *optimistic concurrency control*. DB2 can use optimistic concurrency control to shorten the amount of time that locks are held in the following situations:

- Between consecutive fetch operations
- Between fetch operations and subsequent positioned update or delete operations

Figure 80 and Figure 81 show processing of positioned update and delete operations without optimistic concurrency control and with optimistic concurrency control.

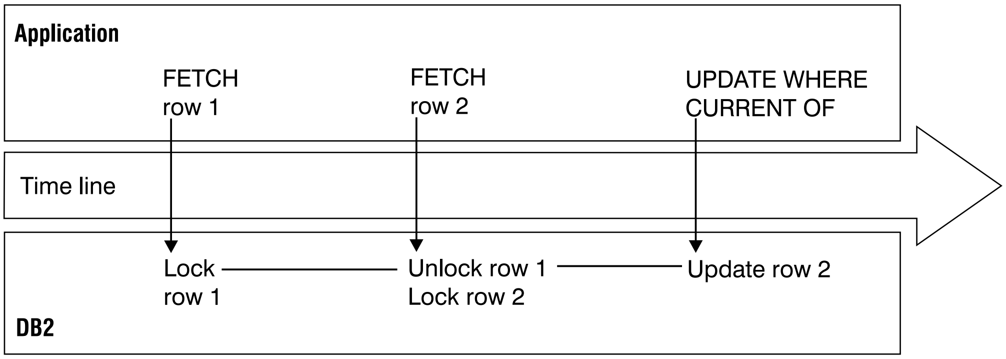


*Figure 80. Positioned updates and deletes without optimistic concurrency control*

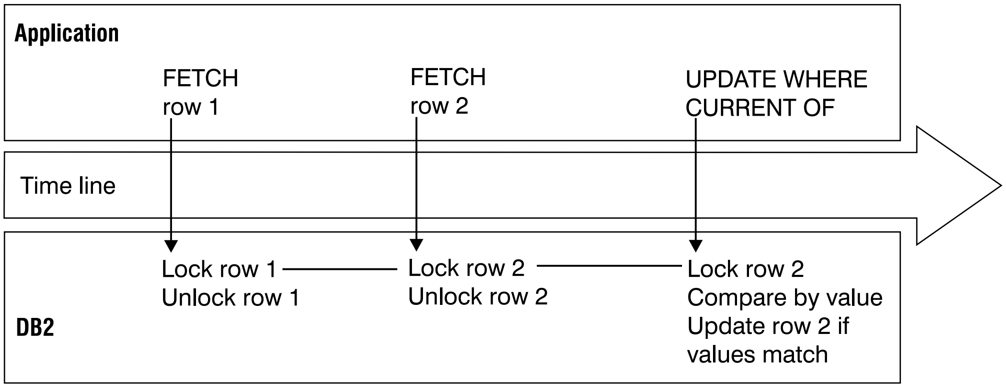


*Figure 81. Positioned updates and deletes with optimistic concurrency control*

Optimistic concurrency control consists of the following steps:

1. When the application requests a fetch operation to position the cursor on a row, DB2 locks that row, executes the FETCH, and releases the lock.
2. When the application requests a positioned update or delete operation on the row, DB2 performs the following steps:
   a. Locks the row.
   b. Reevaluates the predicate to ensure that the row still qualifies for the result table.

c. For columns that are in the result table, compares current values in the row to the values of the row when step 1 was executed. Performs the positioned update or delete operation only if the values match.

**End of Product-sensitive Programming Interface**

**ISOLATION (UR)**

Allows the application to read while acquiring few locks, at the risk of reading uncommitted data. UR isolation applies only to read-only operations: SELECT, SELECT INTO, or FETCH from a read-only result table.

**There is an element of uncertainty about reading uncommitted data.**

**Example**: An application tracks the movement of work from station to station along an assembly line. As items move from one station to another, the application subtracts from the count of items at the first station and adds to the count of items at the second. Assume you want to query the count of items at all the stations, while the application is running concurrently.

What can happen if your query reads data that the application has changed but has not committed?

> If the application subtracts an amount from one record before adding it to another, *the query could miss the amount entirely.*
>
> If the application adds first and then subtracts, *the query could add the amount twice*.

If those situations can occur and are unacceptable, do not use UR isolation.

**Restrictions:** You cannot use UR isolation for the types of statement listed below. If you bind with ISOLATION(UR), and the statement does not specify WITH RR or WITH RS, then DB2 uses CS isolation for:

- INSERT, UPDATE, and DELETE
- Any cursor defined with FOR UPDATE OF

**When can you use uncommitted read (UR)?** You can probably use UR isolation in cases like the following ones:

- **When errors cannot occur**.

  **Example**: A reference table, like a table of descriptions of parts by part number. It is rarely updated, and reading an uncommitted update is probably no more damaging than reading the table 5 seconds earlier. Go ahead and read it with ISOLATION(UR).

  **Example**: The employee table of Spiffy Computer, our hypothetical user. For security reasons, updates can be made to the table only by members of a single department. And that department is also the only one that can query the entire table. It is easy to restrict queries to times when no updates are being made and then run with UR isolation.

- **When an error is acceptable**.

  **Example**: Spiffy wants to do some statistical analysis on employee data. A typical question is, “What is the average salary by sex within education level?” Because reading an occasional uncommitted record cannot affect the averages much, UR isolation can be used.

- **When the data already contains inconsistent information**.

**Example**: Spiffy gets sales leads from various sources. The data is often inconsistent or wrong, and end users of the data are accustomed to dealing with that. Inconsistent access to a table of data on sales leads does not add to the problem.

**Do NOT use uncommitted read (UR):**
**When the computations must balance**
**When the answer must be accurate**
**When you are not sure it can do no damage**

***Plans and packages that use UR isolation:*** Auditors and others might need to determine what plans or packages are bound with UR isolation. For queries that select that information from the catalog, see “What ensures that concurrent users access consistent data?” on page 228.

***Restrictions on concurrent access:*** An application using UR isolation cannot run concurrently with a utility that drains all claim classes. Also, the application must acquire the following locks:

- A special *mass delete lock* acquired in S mode on the target table or table space. A “mass delete” is a DELETE statement without a WHERE clause; that operation must acquire the lock in X mode and thus cannot run concurrently.
- An IX lock on any table space used in the work file database. That lock prevents dropping the table space while the application is running.
- If LOB values are read, LOB locks and a lock on the LOB table space. If the LOB lock is not available because it is held by another application in an incompatible lock state, the UR reader skips the LOB and moves on to the next LOB that satisfies the query.

## The CURRENTDATA option

The CURRENTDATA option has different effects, depending on if access is local or remote:

- For **local** access, the option tells whether the data upon which your cursor is positioned must remain identical to (or “current with”) the data in the local base table. For cursors positioned on data in a work file, the CURRENTDATA option has no effect. This effect only applies to read-only or *ambiguous* cursors in plans or packages bound with CS isolation.

  A cursor is “ambiguous” if DB2 cannot tell whether it is used for update or read-only purposes. If the cursor appears to be used only for read-only, but dynamic SQL could modify data through the cursor, then the cursor is ambiguous. If you use CURRENTDATA to indicate an ambiguous cursor is read-only when it is actually targeted by dynamic SQL for modification, you’ll get an error. See “Problems with ambiguous cursors” on page 687 for more information about ambiguous cursors.
- For a request to a **remote** system, CURRENTDATA has an effect for ambiguous cursors using isolation levels RR, RS, or CS. For ambiguous cursors, it turns block fetching on or off. (Read-only cursors and UR isolation always use block fetch.) Turning on block fetch offers best performance, but it means the cursor is not current with the base table at the remote site.

***Local access:*** Locally, **CURRENTDATA(YES)** means that the data upon which the cursor is positioned cannot change while the cursor is positioned on it. If the cursor is positioned on data in a local base table or index, then the data returned with the cursor is current with the contents of that table or index. If the cursor is positioned on data in a work file, the data returned with the cursor is current only with the contents of the work file; it is not necessarily current with the contents of the underlying table or index.

Figure 82 shows locking with CURRENTDATA(YES).

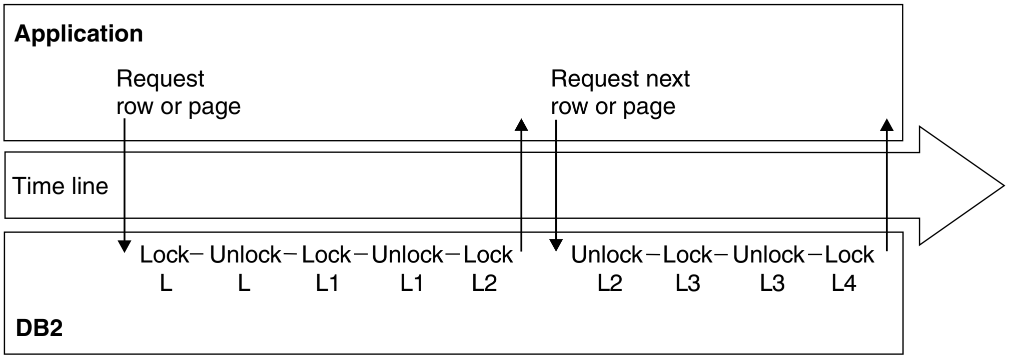


*Figure 82. How an application using isolation CS with CURRENTDATA(YES) acquires locks. This figure shows access to the base table. The L2 and L4 locks are released after DB2 moves to the next row or page. When the application commits, the last lock is released.*

As with work files, if a cursor uses query parallelism, data is not necessarily current with the contents of the table or index, regardless of whether a work file is used. Therefore, for work file access or for parallelism on read-only queries, the CURRENTDATA option has no effect.

If you are using parallelism but want to maintain currency with the data, you have the following options:

- Disable parallelism (Use SET DEGREE = ’1’ or bind with DEGREE(1)).
- Use isolation RR or RS (parallelism can still be used).
- Use the LOCK TABLE statement (parallelism can still be used).

For local access, **CURRENTDATA(NO)** is similar to CURRENTDATA(YES) except for the case where a cursor is accessing a base table rather than a result table in a work file. In those cases, although CURRENTDATA(YES) can guarantee that the cursor and the base table are current, CURRENTDATA(NO) makes no such guarantee.

***Remote access:*** For access to a remote table or index, **CURRENTDATA(YES)** turns off block fetching for ambiguous cursors. The data returned with the cursor is current with the contents of the remote table or index for ambiguous cursors. See “Ensuring block fetch” on page 861 for information about the effect of CURRENTDATA on block fetch.

***Lock avoidance:*** With CURRENTDATA(NO), you have much greater opportunity for avoiding locks. DB2 can test to see if a row or page has committed data on it. If it has, DB2 does not have to obtain a lock on the data at all. Unlocked data is returned to the application, and the data can be changed while the cursor is positioned on the row. (For SELECT statements in which no cursor is used, such as those that return a single row, a lock is not held on the row unless you specify WITH RS or WITH RR on the statement.)

To take the best advantage of this method of avoiding locks, make sure all applications that are accessing data concurrently issue COMMITs frequently.

Figure 83 shows how DB2 can avoid taking locks and Table 96 summarizes the factors that influence lock avoidance.

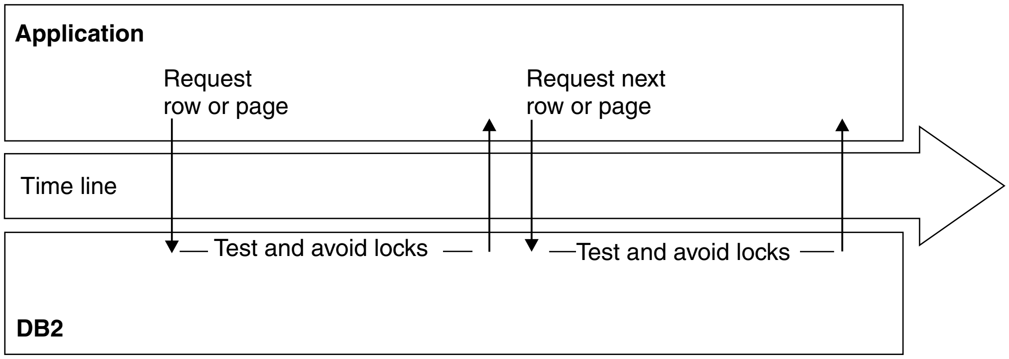


*Figure 83. Best case of avoiding locks using CS isolation with CURRENTDATA(NO). This figure shows access to the base table. If DB2 must take a lock, then locks are released when DB2 moves to the next row or page, or when the application commits (the same as CURRENTDATA(YES)).*

*Table 96. Lock avoidance factors. “Returned data” means data that satisfies the predicate. “Rejected data” is that which does not satisfy the predicate.*

| Isolation | CURRENTDATA | Cursor type | Avoid locks on returned data? | Avoid locks on rejected data? |
|---|---|---|---|---|
| UR | N/A | Read-only | N/A | N/A |
| CS | YES | Read-only | No | Yes |
| | | Updatable | | |
| | | Ambiguous | | |
| | NO | Read-only | Yes | |
| | | Updatable | No | |
| | | Ambiguous | Yes | |
| RS | N/A | Read-only | No | Yes |
| | | Updatable | | |
| | | Ambiguous | | |
| RR | N/A | Read-only | No | No |
| | | Updatable | | |
| | | Ambiguous | | |

***Problems with ambiguous cursors:*** As shown in Table 96, ambiguous cursors can sometimes prevent DB2 from using lock avoidance techniques. However, misuse of an ambiguous cursor can cause your program to receive a -510 SQLCODE:

- The plan or package is bound with CURRENTDATA(NO)
- An OPEN CURSOR statement is performed *before* a dynamic DELETE WHERE CURRENT OF statement against that cursor is prepared
- One of the following conditions is true for the open cursor:
  - Lock avoidance is successfully used on that statement.
  - Query parallelism is used.

– The cursor is distributed, and block fetching is used.

In all cases, it is a good programming technique to eliminate the ambiguity by declaring the cursor with one of the clauses FOR FETCH ONLY or FOR UPDATE OF.

## When plan and package options differ

A plan bound with one set of options can include packages in its package list that were bound with different sets of options. In general, statements in a DBRM bound as a package use the options that the package was bound with, and statements in DBRMs bound to a plan use the options that the plan was bound with.

For example, the plan value for CURRENTDATA has no effect on the packages executing under that plan. If you do not specify a CURRENTDATA option explicitly when you bind a package, the default is CURRENTDATA(YES).

The rules are slightly different for the bind options RELEASE and ISOLATION. The values of those two options are set when the lock on the resource is acquired and usually stay in effect until the lock is released. But a conflict can occur if a statement that is bound with one pair of values requests a lock on a resource that is already locked by a statement that is bound with a different pair of values. DB2 resolves the conflict by resetting each option with the available value that causes the lock to be held for the greatest duration.

If the conflict is between RELEASE(COMMIT) and RELEASE(DEALLOCATE), then the value used is RELEASE(DEALLOCATE).

Table 97 shows how conflicts between isolation levels are resolved. The first column is the existing isolation level, and the remaining columns show what happens when another isolation level is requested by a new application process.

*Table 97. Resolving isolation conflicts*

| | **UR** | **CS** | **RS** | **RR** |
|---|---|---|---|---|
| **UR** | n/a | CS | RS | RR |
| **CS** | CS | n/a | RS | RR |
| **RS** | RS | RS | n/a | RR |
| **RR** | RR | RR | RR | n/a |

## The effect of WITH HOLD for a cursor

For a cursor defined as WITH HOLD, the cursor position is maintained past a commit point. Hence, locks and claims needed to maintain that position are not released immediately, even if they were acquired with ISOLATION(CS) or RELEASE(COMMIT).

For locks and claims needed for cursor position, the rules described above differ as follows:

***Page and row locks:*** If you specify NO on field RELEASE LOCKS on installation panel DSNTIP4, described in “Option to release locks for cursors defined WITH HOLD” on page 673, a page or row lock, if the lock is not successfully avoided through lock avoidance, is held past the commit point. This page or row lock is not necessary for cursor position, but the NO option is provided for compatibility with applications that might rely on this lock. However, an X or U lock is demoted to an S lock at that time. (Because changes have been committed, exclusive control is no longer needed.) After the commit point, the lock is released at the next commit point, provided that no cursor is still positioned on that page or row.

A YES for RELEASE LOCKS means that no data page or row locks are held past commit.

***Table, table space, and DBD locks:*** All necessary locks are held past the commit point. After that, they are released according to the RELEASE option under which they were acquired: for COMMIT, at the next commit point after the cursor is closed; for DEALLOCATE, when the application is deallocated.

***Claims:*** All claims, for any claim class, are held past the commit point. They are released at the next commit point after all held cursors have moved off the object or have been closed.

## Isolation overriding with SQL statements

The information under this heading, up to "The statement LOCK TABLE" on page 690 is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page 1095.

***Function of the WITH clause:*** You can override the isolation level with which a plan or package is bound by the WITH clause on certain SQL statements.

***Example:*** This statement:

```
SELECT MAX(BONUS), MIN(BONUS), AVG(BONUS)
  INTO :MAX, :MIN, :AVG
  FROM DSN8710.EMP
     WITH UR;
```

finds the maximum, minimum, and average bonus in the sample employee table. The statement is executed with uncommitted read isolation, regardless of the value of ISOLATION with which the plan or package containing the statement is bound.

***Rules for the WITH clause:*** The WITH clause:

- Can be used on these statements:
  - Select-statement
  - SELECT INTO
  - Searched delete
  - INSERT from fullselect
  - Searched update
- Cannot be used on subqueries.
- Can specify the isolation levels that specifically apply to its statement. (For example, because WITH UR applies only to read-only operations, you cannot use it on an INSERT statement.)
- Overrides the isolation level for the plan or package only for the statement in which it appears.

***Using KEEP UPDATE LOCKS on the WITH clause:*** You can use the clause KEEP UPDATE LOCKS clause when you specify a SELECT with FOR UPDATE OF. This option is only valid when you use WITH RR or WITH RS. By using this clause, you tell DB2 to acquire an X lock instead of an U or S lock on all the qualified pages or rows.

Here is an example:

```
SELECT ...
 FOR UPDATE OF WITH RS KEEP UPDATE LOCKS;
```

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

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

All X locks are held until the application commits. Although this option can reduce concurrency, it can prevent some types of deadlocks and can better serialize access to data.

## The statement LOCK TABLE

The information under this heading, up to “Claims and drains for concurrency control” on page 695 is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

For information about using LOCK TABLE on an auxiliary table, see “The LOCK TABLE statement” on page 695.

### The purpose of LOCK TABLE

Use the LOCK TABLE statement to override DB2’s rules for choosing initial lock attributes. Two examples are:

```
LOCK TABLE table-name IN SHARE MODE;
LOCK TABLE table-name PART n IN EXCLUSIVE MODE;
```

Executing the statement requests a lock immediately, unless a suitable lock exists already, as described below. The bind option RELEASE determines when locks acquired by LOCK TABLE or LOCK TABLE with the PART option are released.

You can use LOCK TABLE on any table, including auxiliary tables of LOB table spaces. See “The LOCK TABLE statement” on page 695 for information about locking auxiliary tables.

LOCK TABLE has no effect on locks acquired at a remote server.

### When to use LOCK TABLE

The statement is often appropriate for a particularly high-priority application. The statement can improve performance if LOCKMAX disables lock escalation or sets a high threshold for it.

For example, suppose that you intend to execute an SQL statement to change job code 21A to code 23 in a table of employee data. The table is defined with:

- The name PERSADM1.EMPLOYEE_DATA
- LOCKSIZE ROW
- LOCKMAX 0, which disables lock escalation

Because the change affects about 15% of the employees, the statement can require many row locks of mode X. To avoid the overhead for locks, first execute:

```
LOCK TABLE PERSADM1.EMPLOYEE_DATA IN EXCLUSIVE MODE;
```

If EMPLOYEE_DATA is a partitioned table space that is defined with LOCKPART YES, you could choose to lock individual partitions as you update them. The PART option is available only for table spaces defined with LOCKPART YES. See “Effects of table spaces of different types” on page 651 for more information about LOCKPART YES. An example is:

```
LOCK TABLE PERSADM1.EMPLOYEE_DATA PART 1 IN EXCLUSIVE MODE;
```

When the statement is executed, DB2 locks partition 1 with an X lock. The lock has no effect on locks that already exist on other partitions in the table space.

### The effect of LOCK TABLE

Table 98 shows the modes of locks acquired in segmented and nonsegmented table spaces for the SHARE and EXCLUSIVE modes of LOCK TABLE. Auxiliary tables of LOB table spaces are considered nonsegmented table spaces and have the same locking behavior.

*Table 98. Modes of locks acquired by LOCK TABLE. LOCK TABLE on partitions behave the same as nonsegmented table spaces.*

| | | **Segmented Table Space** | |
|---|---|---|---|
| **LOCK TABLE IN** | **Nonsegmented Table Space** | **Table** | **Table Space** |
| EXCLUSIVE MODE | X | X | IX |
| SHARE MODE | S or SIX | S or SIX | IS |

**Note:** The SIX lock is acquired if the process already holds an IX lock. SHARE MODE has no effect if the process already has a lock of mode SIX, U, or X.

# LOB locks

The locking activity for LOBs is described separately from transaction locks because the purpose of LOB locks is different than that of regular transaction locks.

***Terminology:*** A lock that is taken on a LOB value in a LOB table space is called a **LOB lock**.

In this section: The following topics are described:

- “Relationship between transaction locks and LOB locks”
- “Hierarchy of LOB locks” on page 693
- “LOB and LOB table space lock modes” on page 693
- “Duration of locks” on page 693
- “Instances when locks on LOB table space are not taken” on page 694
- “Control of the number of locks” on page 694
- “The LOCK TABLE statement” on page 695
- “The LOCKSIZE clause for LOB table spaces” on page 695

## Relationship between transaction locks and LOB locks

As described in *DB2 Application Programming and SQL Guide*, LOB column values are stored in a different table space, a LOB table space, from the values in the base table. An application that reads or updates a row in a table that contains LOB columns obtains its normal transaction locks on the base table. The locks on the base table also control concurrency for the LOB table space. When locks are not acquired on the base table, such as for ISO(UR), DB2 maintains data consistency by using locks on the LOB table space. Even when locks are acquired on the base table, DB2 still obtains locks on the LOB table space.

DB2 also obtains locks on the LOB table space and the LOB values stored in that LOB table space, but those locks have the following primary purposes:

- To determine whether space from a deleted LOB can be reused by an inserted or updated LOB

Storage for a deleted LOB is not reused until no more readers (including held locators) are on the LOB and the delete operation has been committed.

- To prevent deallocating space for a LOB that is currently being read

  A LOB can be deleted from one application's point-of-view while a reader from another application is reading the LOB. The reader continues reading the LOB because all readers, including those readers that are using uncommitted read isolation, acquire S-locks on LOBs to prevent the storage for the LOB they are reading from being deallocated. That lock is held until commit. A held LOB locator or a held cursor cause the LOB lock and LOB table space lock to be held past commit.

In summary, the main purpose of LOB locks is for managing the space used by LOBs and to ensure that LOB readers do not read partially updated LOBs. Applications need to free held locators so that the space can be reused.

Table 99 shows the relationship between the action that is occurring on the LOB value and the associated LOB table space and LOB locks that are acquired.

*Table 99. Locks that are acquired for operations on LOBs. This table does not account for gross locks that can be taken because of LOCKSIZE TABLESPACE, the LOCK TABLE statement, or lock escalation.*

| Action on LOB value | LOB table space lock | LOB lock | Comment |
|---|---|---|---|
| Read (including UR) | IS | S | Prevents storage from being reused while the LOB is being read or while locators are referencing the LOB |
| Insert | IX | X | Prevents other processes from seeing a partial LOB |
| Delete | IS | S | To hold space in case the delete is rolled back. (The X is on the base table row or page.) Storage is not reusable until the delete is committed and no other readers of the LOB exist. |
| Update | IS->IX | Two LOB locks: an S-lock for the delete and an X-lock for the insert. | Operation is a delete followed by an insert. |
| Update the LOB to null or zero-length | IS | S | No insert, just a delete. |
| Update a null or zero-length LOB to a value | IX | X | No delete, just an insert. |

***ISOLATION(UR) or ISOLATION(CS):*** When an application is reading rows using uncommitted read or lock avoidance, no page or row locks are taken on the base table. Therefore, these readers must take an S LOB lock to ensure that they are not reading a partial LOB or a LOB value that is inconsistent with the base row.

## Hierarchy of LOB locks

Just as page locks (or row locks) and table space locks have a hierarchical relationship, LOB locks and locks on LOB table spaces have a hierarchical relationship. (See Figure 75 on page 651 for a picture of the hierarchical relationship.) If the LOB table space is locked with a gross lock, then LOB locks are not acquired. In a data sharing environment, the lock on the LOB table space is used to determine whether the lock on the LOB must be propagated beyond the local IRLM.

## LOB and LOB table space lock modes

### Modes of LOB locks

The following LOB lock modes are possible:

**S (SHARE)**
The lock owner and any concurrent processes can read, update, or delete the locked LOB. Concurrent processes can acquire an S lock on the LOB. The purpose of the S lock is to reserve the space used by the LOB.

**X (EXCLUSIVE)**
The lock owner can read or change the locked LOB. Concurrent processes cannot access the LOB.

### Modes of LOB table space locks

The following locks modes are possible on the LOB table space:

**IS (INTENT SHARE)**
The lock owner can update LOBs to null or zero-length, or read or delete LOBs in the LOB table space. Concurrent processes can both read and change LOBs in the same table space. The lock owner acquires a LOB lock on any data that it reads or deletes.

**IX (INTENT EXCLUSIVE)**
The lock owner and concurrent processes can read and change data in the LOB table space. The lock owner acquires a LOB lock on any data it accesses.

**S (SHARE)**
The lock owner and any concurrent processes can read and delete LOBs in the LOB table space. The lock owner does not need LOB locks.

**SIX (SHARE with INTENT EXCLUSIVE)**
The lock owner can read and change data in the LOB table space. If the lock owner is inserting (INSERT or UPDATE), the lock owner obtains a LOB lock. Concurrent processes can read or delete data in the LOB table space (or update to a null or zero-length LOB).

**X (EXCLUSIVE)**
The lock owner can read or change LOBs in the LOB table space. The lock owner does not need LOB locks. Concurrent processes cannot access the data.

## Duration of locks

### Duration of locks on LOB table spaces

Locks on LOB table spaces are acquired when they are needed; that is, the ACQUIRE option of BIND has no effect on when the table space lock on the LOB table space is taken. The table space lock is released according to the value specified on the RELEASE option of BIND (except when a cursor is defined WITH HOLD or if a held LOB locator exists).

### Duration of LOB locks

Locks on LOBs are taken when they are needed and are usually released at commit. However, if that LOB value is assigned to a LOB locator, the S lock remains until the application commits.

If the application uses HOLD LOCATOR, the LOB lock is not freed until the first commit operation after a FREE LOCATOR statement is issued, or until the thread is deallocated.

***A note about held cursors:*** If a cursor is defined WITH HOLD, LOB locks are held through commit operations.

| ***A note about INSERT with fullselect:*** Because LOB locks are held until commit and because locks are put on each LOB column in both a source table and a target table, it is possible that a statement such as an INSERT with a fullselect that involves LOB columns can accumulate many more locks than a similar statement that does not involve LOB columns. To prevent system problems caused by too many locks, you can:

- Ensure that you have lock escalation enabled for the LOB table spaces that are involved in the INSERT. In other words, make sure that LOCKMAX is non-zero for those LOB table spaces.
- Alter the LOB table space to change the LOCKSIZE to TABLESPACE before executing the INSERT with fullselect.
- Increase the LOCKMAX value on the table spaces involved and ensure that the user lock limit is sufficient.
- Use LOCK TABLE statements to lock the LOB table spaces. (Locking the auxiliary table that is contained in the LOB table space locks the LOB table space.)

## Instances when locks on LOB table space are not taken

A lock might not be acquired on a LOB table space at all. For example, if a row is deleted from a table and the value of the LOB column is null, the LOB table space associated with that LOB column is not locked. DB2 does not access the LOB table space if the application:

- Selects a LOB that is null or zero length
- Deletes a row where the LOB is null or zero length
- Inserts a null or zero length LOB
- Updates a null or zero-length LOB to null or zero-length

## Control of the number of locks

This section describes how you can control the number of LOB locks that are taken.

### Controlling the number of LOB locks that are acquired for a user

LOB locks are counted toward the total number of locks allowed per user. Control this number by the value you specify on the LOCKS PER USER field of installation panel DSNTIPJ. The number of LOB locks that are acquired during a unit of work is reported in IFCID 0020.

#### Controlling LOB lock escalation

As with any table space, use the LOCKMAX clause of the CREATE or ALTER TABLESPACE statement to control the number of LOB locks that are acquired within a particular LOB table space before the lock is escalated. See “LOCKMAX clause of CREATE and ALTER TABLESPACE” on page 672 for more information. When the number of LOB locks reaches the maximum you specify in the LOCKMAX clause, the LOB locks escalate to a gross lock on the LOB table space, and the LOB locks are released.

Information about LOB locks and lock escalation is reported in IFCID 0020.

### The LOCK TABLE statement

“The statement LOCK TABLE” on page 690 describes how and why you might use a LOCK TABLE statement on a table. The reasons for using LOCK TABLE on an auxiliary table are somewhat different than that for regular tables.

- You can use LOCK TABLE to control the number of locks acquired on the auxiliary table.
- You can use LOCK TABLE IN SHARE MODE to prevent other applications from inserting LOBs.

  With auxiliary tables, LOCK TABLE IN SHARE MODE does not prevent any changes to the auxiliary table. The statement does prevent LOBs from being inserted into the auxiliary table, but it does not prevent deletes. Updates are generally restricted also, except where the LOB is updated to a null value or a zero-length string.
- You can use LOCK TABLE IN EXCLUSIVE MODE to prevent other applications from accessing LOBs.

  With auxiliary tables, LOCK TABLE IN EXCLUSIVE MODE also prevents access from uncommitted readers.
- Either statement eliminates the need for lower-level LOB locks.

### The LOCKSIZE clause for LOB table spaces

The LOCKSIZE TABLE, PAGE, and ROW options are not valid for LOB table spaces. The other options act as follows:

**LOCKSIZE TABLESPACE**
A process acquires no LOB locks.

**LOCKSIZE ANY**
DB2 chooses the size of the lock. For a LOB table space, this is usually LOCKSIZE LOB.

**LOCKSIZE LOB**
If LOBs are accessed, a process acquires LOB locks and the necessary LOB table space locks (IS or IX).

## Claims and drains for concurrency control

DB2 utilities, commands, and some ALTER, CREATE, and DROP statements can take over access to some objects independently of any transaction locks that are held on the object.

### Objects subject to takeover

DB2 utilities, commands, and some ALTER, CREATE, and DROP statements can take over access to the following table and index spaces:

- Simple and segmented table spaces
- Partitions of table spaces
- LOB table spaces
- Nonpartitioned index spaces
- Partitions of index spaces
- Logical partitions of nonpartitioning index

The effects of those takeovers are described in the following sections:

- “Definition of claims and drains”
- “Usage of drain locks” on page 697
- “Utility locks on the catalog and directory” on page 697
- “Compatibility of utilities” on page 698
- “Concurrency during REORG” on page 699
- “Utility operations with nonpartitioning indexes” on page 700

## Definition of claims and drains

### Definition

A *claim* is a notification to DB2 that an object is being accessed.

### Example

When an application first accesses an object, within a unit of work, it makes a claim on the object. It releases the claim at the next commit point.

### Effects of a claim

Unlike a transaction lock, a claim normally does not persist past the commit point. To access the object in the next unit of work, the application must make a new claim.

However, there is an exception. If a cursor defined with the clause WITH HOLD is positioned on the claimed object, the claim is not released at a commit point. For more about cursors defined as WITH HOLD, see “The effect of WITH HOLD for a cursor” on page 688.

A claim indicates to DB2 that there is activity on or interest in a particular page set or partition. Claims prevent drains from occurring until the claim is released.

### Three classes of claims

| Claim Class | Actions Allowed |
|---|---|
| **Write** | Reading, updating, inserting, and deleting |
| **Repeatable read** | Reading only, with repeatable read (RR) isolation |
| **Cursor stability read** | Reading only, with read stability (RS), cursor stability (CS), or uncommitted read (UR) isolation |

### Definition

A *drain* is the action of taking over access to an object by preventing new claims and waiting for existing claims to be released.

### Example

A utility can drain a partition when applications are accessing it.

### Effects of a drain

The drain quiesces the applications by allowing each one to reach a commit point, but preventing any of them, or any other applications, from making a new claim. When no more claims exist, the process that drains (the drainer) controls access to the drained object. The applications that were drained can still hold transaction locks on the drained object, but they cannot make new claims until the drainer has finished.

### Claim classes drained

A drainer does not always need complete control. It could drain:

- Only the write claim class
- Only the repeatable read claim class
- All claim classes

For example, the CHECK INDEX utility needs to drain only writers from an index space and its associated table space. RECOVER, however, must drain all claim classes from its table space. The REORG utility can drain either writers (with DRAIN WRITERS) or all claim classes (with DRAIN ALL).

## Usage of drain locks

### Definition

A *drain lock* prevents conflicting processes from trying to drain the same object at the same time. Processes that drain only writers can run concurrently; but a process that drains all claim classes cannot drain an object concurrently with any other process. To drain an object, a drainer first acquires one or more drain locks on the object, one for each claim class it needs to drain. When the locks are in place, the drainer can begin at the next commit point or after the release of all held cursors.

A drain lock also prevents new claimers from accessing an object while a drainer has control of it.

### Types of drain locks

Three types of drain locks on an object correspond to the three claim classes:

- Write
- Repeatable read
- Cursor stability read

In general, after an initial claim has been made on an object by a user, no other user in the system needs a drain lock. When the drain lock is granted, no drains on the object are in process for the claim class needed, and the claimer can proceed.[10]

## Utility locks on the catalog and directory

When the target of a utility is an object in the catalog or directory, such as a catalog table, the utility either drains or claims the object.

When the target is a user-defined object, the utility claims or drains it but also uses the directory and, perhaps, the catalog; for example, to check authorization. In those cases, the utility uses transaction locks on catalog and directory tables. It acquires those locks in the same way as an SQL transaction does.

[10] The claimer of an object requests a drain lock in two exceptional cases:

- A drain on the object is in process for the claim class needed. In this case, the claimer waits for the drain lock.
- The claim is the first claim on an object before its data set has been physically opened. Here, acquiring the drain lock ensures that no exception states prohibit allocating the data set.

When the claimer gets the drain lock, it makes its claim and releases the lock before beginning its processing.

***The UTSERIAL lock:*** Access to the SYSUTILX table space in the directory is controlled by a unique lock called UTSERIAL. A utility must acquire the UTSERIAL lock to read or write in SYSUTILX, whether SYSUTILX is the target of the utility or is used only incidentally.

# Compatibility of utilities

## Definition

Two utilities are considered *compatible* if they do not need access to the same object at the same time in incompatible modes.

## Compatibility rules

The concurrent operation of two utilities is not typically controlled by either drain locks or transaction locks, but merely by a set of compatibility rules.

Before a utility starts, it is checked against all other utilities running on the same target object. The utility starts only if all the others are compatible.

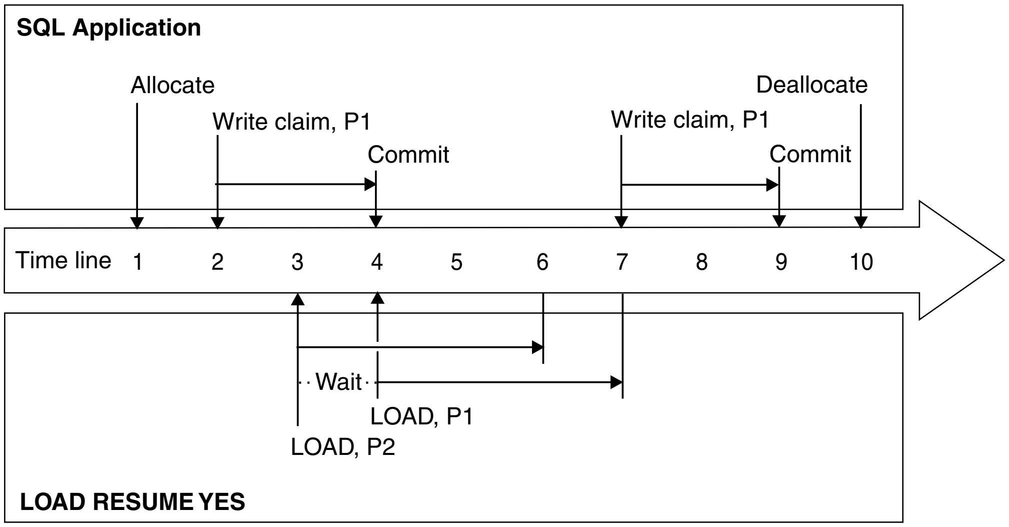
The check for compatibility obeys the following rules:

- The check is made for each target object, but only for target objects. Typical utilities access one or more table spaces or indexes, but if two utility jobs use none of the same target objects, the jobs are always compatible.

  An exception is a case in which one utility must update a catalog or directory table space that is not the direct target of the utility. For example, the LOAD utility on a user table space updates DSNDB06.SYSCOPY. Therefore, other utilities that have DSNDB06.SYSCOPY as a target might not be compatible.
- Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions in the same table or index space are compatible.
- When two utilities access the same target object, their most restrictive access modes determine whether they are compatible. For example, if utility job 1 reads a table space during one phase and writes during the next, it is considered a writer. It cannot start concurrently with utility 2, which allows only readers on the table space. (Without this restriction, utility 1 might start and run concurrently with utility 2 for one phase; but then it would fail in the second phase, because it could not become a writer concurrently with utility 2.)

For details on which utilities are compatible, refer to each utility's description in *DB2 Utility Guide and Reference*.

Figure 84 on page 699 illustrates how SQL applications and DB2 utilities can operate concurrently on separate partitions of the same table space.



**Time** **Event**

**t1** An SQL application obtains a transaction lock on every partition in the table space. The duration of the locks extends until the table space is deallocated.

**t2** The SQL application makes a write claim on data partition 1 and index partition 1.

**t3** The LOAD jobs begin draining all claim classes on data partitions 1 and 2 and index partitions 1 and 2. LOAD on partition 2 operates concurrently with the SQL application on partition 1. LOAD on partition 1 waits.

**t4** The SQL application commits, releasing its write claims on partition 1. LOAD on partition 1 can begin.

**t6** LOAD on partition 2 completes.

**t7** LOAD on partition 1 completes, releasing its drain locks. The SQL application (if it has not timed out) makes another write claim on data partition 1.

**t10** The SQL application deallocates the table space and releases its transaction locks.

*Figure 84. SQL and utility concurrency. Two LOAD jobs execute concurrently on two partitions of a table space*

## Concurrency during REORG

If you are getting deadlocks when you use REORG with the SHRLEVEL CHANGE option, run the REORG utility with the DRAIN ALL option. The default is DRAIN WRITERS, which is done in the log phase. The specification of DRAIN ALL indicates that both writers and readers will be drained when the MAXRO threshold is reached. The DRAIN ALL option should be considered in environments where a lot of update activity occurs during the log phase. With this specification, there is no need for a subsequent drain in the switch phase.

When multiple REORG utility jobs with the SHRLEVEL CHANGE or REFERENCE and the FASTSWITCH options run against separate partitions of the same partitioned table space, some of the utilities might fail with reason code 00E40012. This code, which indicates the unavailability of the database descriptor block (DBD) is caused by multiple utilities arriving at the SWITCH phase simultaneously. The switch phase times out if it cannot acquire the DBD within the timeout period specified by the UTILITY TIMEOUT field on installation panel DSNTIPI. Increase the value of the installation parameter to alleviate the problem.

### Utility operations with nonpartitioning indexes

In the example of Figure 84 on page 699, two LOAD jobs execute concurrently on different partitions of the same table space. When the jobs proceed to build the partitioning index, they operate on different partitions of the index and can again operate concurrently.

But in a nonpartitioning index, an entry can refer to any partition in the underlying table space. DB2 can process a set of entries of a nonpartitioning index that all refer to a single partition and achieve the same results as for a partition of a partitioning index. (Such a set of entries is called a *logical partition* of the index.)

## Monitoring of DB2 locking

If you have problems with suspensions, timeouts, or deadlocks, you will want to monitor DB2's use of locks.

Use the DB2 command DISPLAY DATABASE to find out what locks are held or waiting at any moment on any table space, partition, or index. The report can include claims and drain locks on logical partitions of indexes. For an example, see “Monitoring databases” on page 269.

Use EXPLAIN to monitor the locks required by a particular SQL statement, or all the SQL in a particular plan or package, and see “Using EXPLAIN to tell which locks DB2 chooses”.

Use the statistics trace to monitor the system-wide use of locks, the accounting trace to monitor locks used by a particular application process, and see “Using the statistics and accounting traces to monitor locking” on page 701.

For an example of resolving a particular locking problem, see “Analyzing a concurrency scenario” on page 702.

### Using EXPLAIN to tell which locks DB2 chooses

The information under this heading, up to “Using the statistics and accounting traces to monitor locking” on page 701, is Product-sensitive Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

***Procedure:***

1. Use the EXPLAIN statement, or the EXPLAIN option of the BIND and REBIND subcommands, to determine which modes of table and table space locks DB2 initially assigns for an SQL statement. Follow the instructions under “Obtaining PLAN_TABLE information from EXPLAIN” on page 790. (EXPLAIN does not return information about the locks acquired on LOB table spaces.)
2. EXPLAIN stores its results in a table called PLAN_TABLE. To review the results, query PLAN_TABLE. After running EXPLAIN, each row of PLAN_TABLE describes the processing for a single table, either one named explicitly in the SQL statement that is being explained or an intermediate table that DB2 has to

create. The column TSLOCKMODE of PLAN_TABLE shows an initial lock mode for that table. The lock mode applies to the table or the table space, depending on the value of LOCKSIZE and whether the table space is segmented or nonsegmented.

3. In Table 100, find what table or table space lock is used and whether page or row locks are used also, for the particular combination of lock mode and LOCKSIZE you are interested in.

***For statements executed remotely:*** EXPLAIN gathers information only about data access in the DBMS where the statement is run or the bind operation is carried out. To analyze the locks obtained at a remote DB2 location, you must run EXPLAIN at that location. For more information on running EXPLAIN, and a fuller description of PLAN_TABLE, see “Chapter 33. Using EXPLAIN to improve SQL performance” on page 789.

*Table 100. Which locks DB2 chooses. N/A = Not applicable; Yes = Page or row locks are acquired; No = No page or row locks are acquired.*

| | **Lock mode from EXPLAIN** | | | | |
|---|---|---|---|---|---|
| **Table space structure** | **IS** | **S** | **IX** | **U** | **X** |
| For nonsegmented table spaces: | | | | | |
| Table space lock acquired is: | IS | S | IX | U | X |
| Page or row locks acquired? | Yes | No | Yes | No | No |
| **Note:** For partitioned table spaces defined with LOCKPART YES and for which selective partition locking is used, the lock mode applies only to those partitions that are locked. Lock modes for LOB table spaces are not reported with EXPLAIN. | | | | | |
| For segmented table spaces with: | | | | | |
| LOCKSIZE ANY, ROW, or PAGE | | | | | |
| Table space lock acquired is: | IS | IS | IX | n/a | IX |
| Table lock acquired is: | IS | S | IX | n/a | X |
| Page or row locks acquired? | Yes | No | Yes | No | No |
| LOCKSIZE TABLE | | | | | |
| Table space lock acquired is: | n/a | IS | n/a | IX | IX |
| Table lock acquired is: | n/a | S | n/a | U | X |
| Page or row locks acquired? | No | No | No | No | No |
| LOCKSIZE TABLESPACE | | | | | |
| Table space lock acquired is: | n/a | S | n/a | U | X |
| Table lock acquired is: | n/a | n/a | n/a | n/a | n/a |
| Page or row locks acquired? | No | No | No | No | No |

## Using the statistics and accounting traces to monitor locking

The statistics and accounting trace records contain information on locking. The IBM licensed program, DATABASE 2 Performance Monitor (DB2 PM), provides one way to view the trace results. Figure 85 on page 702 contains extracts from the DB2 PM reports Accounting Trace and Statistics Trace. Each of those corresponds to a single DB2 trace record. (Details of those reports are subject to change without notification from DB2 and are available in the appropriate DB2 PM documentation). As the figure shows:

- Statistics Trace tells how many suspensions, deadlocks, timeouts, and lock escalations occur in the trace record.
- Accounting Trace gives the same information for a particular application. It also shows the maximum number of concurrent page locks held and acquired during the trace. Review applications with a large number to see if this value can be

lowered. This number is the basis for the proper setting of LOCKS PER USER and, indirectly, LOCKS PER TABLE(SPACE).

***Recommendations:*** Check the results of the statistics and accounting traces for the following possibilities:

- Lock escalations are generally undesirable and are caused by processes that use a large number of page, row, or LOB locks. In some cases, it is possible to improve system performance by using table or table space locks.
- Timeouts can be caused by a small value of RESOURCE TIMEOUT. If there are many timeouts, check whether a low value for RESOURCE TIMEOUT is causing them. Sometimes the problem suggests a need for some change in database design.

```
                                                                ||
LOCKING ACTIVITY            QUANTITY  /MINUTE  /THREAD  /COMMIT ||  LOCKING                TOTAL
--------------------------  --------  -------  -------  ------- ||  -------------------  --------
SUSPENSIONS (ALL)                  2     1.28     1.00     0.40 ||  TIMEOUTS                    0
SUSPENSIONS (LOCK ONLY)            2     1.28     1.00     0.40 ||  DEADLOCKS                   0
SUSPENSIONS (IRLM LATCH)           0     0.00     0.00     0.00 ||  ESCAL.(SHARED)              0
SUSPENSIONS (OTHER)                0     0.00     0.00     0.00 ||  ESCAL.(EXCLUS)              0
                                                                ||  MAX PG/ROW LCK HELD         2
TIMEOUTS                           0     0.00     0.00     0.00 ||  LOCK REQUEST                8
DEADLOCKS                          1     0.64     0.50     0.20 ||  UNLOCK REQUEST              2
                                                                ||  QUERY REQUEST               0
LOCK REQUESTS                     17    10.92     8.50     3.40 ||  CHANGE REQUEST              5
UNLOCK REQUESTS                   12     7.71     6.00     2.40 ||  OTHER REQUEST               0
QUERY REQUESTS                     0     0.00     0.00     0.00 ||  LOCK SUSPENSIONS            1
CHANGE REQUESTS                    5     3.21     2.50     1.00 ||  IRLM LATCH SUSPENS.         0
OTHER REQUESTS                     0     0.00     0.00     0.00 ||  OTHER SUSPENSIONS           0
                                                                ||  TOTAL SUSPENSIONS           1
LOCK ESCALATION (SHARED)           0     0.00     0.00     0.00 ||
LOCK ESCALATION (EXCLUSIVE)        0     0.00     0.00     0.00 ||  DRAIN/CLAIM      TOTAL
                                                                ||  ------------  --------
DRAIN REQUESTS                     0     0.00     0.00     0.00 ||  DRAIN REQST          0
DRAIN REQUESTS FAILED              0     0.00     0.00     0.00 ||  DRAIN FAILED         0
CLAIM REQUESTS                     7     4.50     3.50     1.40 ||  CLAIM REQST          4
CLAIM REQUESTS FAILED              0     0.00     0.00     0.00 ||  CLAIM FAILED         0
```

*Figure 85. Locking activity blocks from statistics trace and accounting trace*

## Analyzing a concurrency scenario

The concurrency problem analyzed in this section illustrates the approach described in “A general approach to problem analysis in DB2” on page 533. It follows the CONCURRENCY PROBLEM branch of Figure 57 on page 535 and makes use of DB2 PM reports. In DB2 PM, a report titled “Trace” corresponds to a single DB2 trace record; a report titled “Report” summarizes information from several trace records. This scenario makes use of:

- Accounting Report - Long
- Locking Report - Suspension
- Locking Report - Lockout
- Locking Trace - Lockout

The following section includes a description of the scenario, an explanation of how each report helps to analyze the situation, and some general information about corrective approaches.

## Scenario description

An application, which has recently been moved into production, is experiencing timeouts. Other applications have not been significantly affected in this example.

To investigate the problem, determine a period when the transaction is likely to time out. When that period begins:

1. Start the GTF.
2. Start the DB2 accounting classes 1, 2, and 3 to GTF to allow for the production of DB2 PM accounting reports.
3. Stop GTF and the traces after about 15 minutes.
4. Produce and analyze the DB2 PM Accounting Report - Long.
5. Use the DB2 performance trace selectively for detailed problem analysis.

In some cases, the initial and detailed stages of tracing and analysis presented in this chapter can be consolidated into one. In other cases, the detailed analysis might not be required at all.

To analyze the problem, generally start with Accounting Report - Long. (If you have enough information from program and system messages, you can skip this first step.)

## Accounting report

Figure 86 on page 704 shows a portion of Accounting Report - Long.

```
PLANNAME: PU22301

AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT    HIGHLIGHTS  D
------------  ----------  ----------  ----------    --------------------  ------------  --------    --------------------------
                  A           B                                                 C
ELAPSED TIME  5:03.57540  5:03.38330         N/P    LOCK/LATCH(DB2+IRLM)   5:03.277805      0.90    #OCCURRENCES    :        2
 NON-NESTED     0.209291    0.032280         N/A    SYNCHRON. I/O             0.000000      0.00    #ALLIEDS        :        2
 STORED PROC  5:03.36611  5:03.35102         N/A     DATABASE I/O             0.000000      0.00    #ALLIEDS DISTRIB:        0
 UDF            0.000000    0.000000         N/A     LOG WRITE I/O            0.000000      0.00    #DBATS          :        0
 TRIGGER        0.000000    0.000000         N/A    OTHER READ I/O            0.000000      0.00    #DBATS DISTRIB. :        0
                                                    OTHER WRTE I/O            0.000000      0.00    #NO PROGRAM DATA:        0
CPU TIME        0.046199    0.021565         N/P    SER.TASK SWTCH            0.082205      5.00    #NORMAL TERMINAT:        2
 AGENT          0.046199    0.021565         N/P     UPDATE COMMIT            0.013300      0.58    #ABNORMAL TERMIN:        0
  NON-NESTED    0.010858    0.000654         N/A     OPEN/CLOSE               0.041102      3.20    #CP/X PARALLEL  :        0
  STORED PROC   0.035241    0.020911         N/A     SYSLGRNG REC             0.014102      0.65    #IO PARALLELISM :        0
  UDF           0.000000    0.000000         N/A     EXT/DEL/DEF              0.006918      0.31    #INCREMENT. BIND:        0
  TRIGGER       0.000000    0.000000         N/A     OTHER  SERVICE           0.006783      0.26    #COMMITS        :        2
 PAR.TASKS      0.000000    0.000000         N/A    ARC.LOG(QUIES)            0.000000      0.00    #ROLLBACKS      :        1
                                                    ARC.LOG READ              0.000000      0.00    #SVPT REQUESTS  :        0
SUSPEND TIME         N/A  5:03.36001         N/A    STOR.PRC SCHED            0.000000      0.00    #SVPT RELEASE   :        0
 AGENT               N/A  5:03.36001         N/A    UDF SCHEDULE              0.000000      0.10    #SVPT ROLLBACK  :        0
 PAR.TASKS           N/A    0.000000         N/A    DRAIN LOCK                0.000000      0.00    MAX SQL CASC LVL:        1
                                                    CLAIM RELEASE             0.000000      0.00    UPDATE/COMMIT   :     0.00
NOT ACCOUNT.         N/A    0.001725         N/A    PAGE LATCH                0.000000      0.00    SYNCH I/O AVG.  :     0.00
DB2 ENT/EXIT         N/A        5.00         N/A    NOTIFY MSGS.              0.000000      0.00
EN/EX-STPROC         N/A        0.00         N/A    GLOBAL CONT.              0.000000      0.00
EN/EX-UDF            N/A        0.00         N/P    FORCE-AT-COMMIT           0.000000      0.00
DCAPT.DESCR.         N/A         N/A         N/P    ASYNCH IXL REQUESTS       0.000000      0.00
LOG EXTRACT.         N/A         N/A         N/P    TOTAL CLASS 3          5:03.366610      6.00

SQL DML   AVERAGE    TOTAL    SQL DCL          TOTAL    SQL DDL    CREATE   DROP  ALTER    LOCKING                AVERAGE    TOTAL
-------- -------- --------    -------------- --------    ---------- ------ ------ ------    --------------------- -------- --------
SELECT       0.00        0    LOCK TABLE            0    TABLE           0      0      0    TIMEOUTS                  1.00        2
INSERT       0.00        0    GRANT                 0    TEMP TABLE      0    N/A    N/A    DEADLOCKS                 0.00        0
UPDATE       0.00        0    REVOKE                0    AUX TABLE       0    N/A    N/A    ESCAL.(SHARED)            0.00        0
DELETE       0.00        0    SET CURR.SQLID        0    INDEX           0      0      0    ESCAL.(EXCLUS)            0.00        0
                              SET HOST VAR.         0    TABLESPACE      0      0      0    MAX PG/ROW LOCKS HELD     1.00        1
DESCRIBE     0.00        0    SET CUR.DEGREE        0    DATABASE        0      0      0    LOCK REQUEST              0.00        0
DESC.TBL     0.00       10    SET RULES             0    STOGROUP        0      0      0    UNLOCK REQUEST            0.00        0
PREPARE      0.00        0    SET CURR.PATH         0    SYNONYM         0      0    N/A    QUERY REQUEST             0.00        0
OPEN         1.00        2    SET CURR.PREC         0    VIEW            0      0    N/A    CHANGE REQUEST            1.00        2
FETCH        0.00        0    CONNECT TYPE 1        0    ALIAS           0      0    N/A    OTHER REQUEST             0.00        0
CLOSE        0.00        0    CONNECT TYPE 2        6    PACKAGE       N/A      0    N/A    LOCK SUSPENSIONS          1.00        2
                              SET CONNECTION        0    PROCEDURE       0      0      0    IRLM LATCH SUSPENSIONS    0.00        0
                              RELEASE               0    FUNCTION        0      0      0    OTHER SUSPENSIONS         0.00        0
DML-ALL      1.00        2    CALL                  0    TRIGGER         0      0    N/A    TOTAL SUSPENSIONS         1.00        2
                              ASSOC LOCATORS        0    DIST TYPE       0      0    N/A
                              ALLOC CURSOR          0
                              HOLD LOCATOR          0    TOTAL           0      0      0
                              FREE LOCATOR          0    RENAME TBL      0
                              DCL-ALL               0    COMMENT ON      0
                                                         LABEL ON        0

⋮
```

*Figure 86. Excerpt from Accounting Report —long*

Accounting Report - Long shows the average elapsed times and the average number of suspensions per plan execution. In Figure 86:

- The class 1 average elapsed time **A** (AET) is 5 minutes, 3.575 seconds (rounded). The class 2 times show that 5 minutes, 3.383 seconds **B** of that are spent in DB2; the rest is spent in the application.
- The class 2 AET is spent mostly in lock or latch suspensions (LOCK/LATCH **C** is 5 minutes, 3.278 seconds).
- The HIGHLIGHTS section **D** of the report (upper right) shows #OCCURRENCES as 2; that is the number of accounting (IFCID 3) records.

## Lock suspension

To prepare for Locking Report - Suspension, start DB2 performance class 6 to GTF. Because that class traces only suspensions, it does not significantly reduce performance. Figure 87 shows the DB2 PM Locking Report - Suspension.

⋮

```
                                                              --SUSPEND REASONS-- ---------- R E S U M E   R E A S O N S ------
PRIMAUTH      --- L O C K   R E S O U R C E --- TOTAL    LOCAL  GLOB.  S.NFY ---- NORMAL ----  TIMEOUT/CANCEL  --- DEADLOCK --
 PLANNAME     TYPE      NAME                    SUSPENDS LATCH  IRLMQ  OTHER NMBR         AET NMBR         AET NMBR        AET
------------- --------- ----------------------- -------- -----  -----  ----- ---- ----------- ---- ----------- ---- ----------
FPB
 PARALLEL     PARTITION DB  =PARADABA                  2     2      0      0    0         N/C    2  303.277805    0        N/C
                        OB  =TAB1TS                          0      0      0
                        PART= 1
LOCKING REPORT COMPLETE
```

*Figure 87. Portion of DB2 PM Locking Report - Suspension*

This report shows:

- Which plans are suspended, by plan name within primary authorization ID. For statements bound to a package, see the information about the plan that executes the package.
- What IRLM requests and which lock types are causing suspensions.
- Whether suspensions are normally resumed or end in timeouts or deadlocks.
- What the average elapsed time (AET) per suspension is.

The report also shows the reason for the suspensions:

| **Reason** | **Includes...** |
|---|---|
| **LOCAL** | Contention for a local resource |
| **LATCH** | Contention for latches within IRLM (with brief suspension) |
| **GLOB.** | Contention for a global resource |
| **IRLMQ** | An IRLM queued request |
| **S.NFY** | Intersystem message sending |
| **OTHER** | Page latch or drain suspensions, suspensions because of incompatible retained locks in data sharing, or a value for service use. |

The list above shows only the first reason for a suspension. When the original reason is resolved, the request could remain suspended for a second reason.

Each suspension results in either a normal resume, a timeout, or a deadlock.

The report shows that the suspension causing the delay involves access to partition 1 of table space PARADABA.TAB1TS by plan PARALLEL. Two LOCAL suspensions time out after an average of 5 minutes, 3.278 seconds (303.278 seconds).

## Lockout report

Figure 88 on page 706 shows the DB2 PM Locking Report - Lockout. This report shows that plan PARALLEL contends with the plan DSNESPRR. It also shows that contention is occurring on partition 1 of table space PARADABA.TAB1TS.

```
PRIMAUTH           --- L O C K   R E S O U R C E ---                      --------------- A G E N T S --------------
 PLANNAME          TYPE      NAME                    TIMEOUTS DEADLOCKS   MEMBER    PLANNAME  CONNECT   CORRID         HOLDER WAITER
------------------ --------- ----------------------- -------- ---------   --------  --------- --------  ------------   ------ ------
FPB
 PARALLEL          PARTITION DB  =PARADABA                  2         0   N/P       DSNESPRR  TSO       EOA                 2      0
                             OB  =TAB1TS
                             PART= 1
                   ** LOCKOUTS FOR PARALLEL   **            2         0
```

Figure 88. Portion of DB2 PM Locking Report - Lockout

## Lockout trace

Figure 89 shows the DB2 PM Locking Trace - Lockout report.

For each contender, this report shows the database object, lock state (mode), and duration for each contention for a transaction lock.

```
⋮

PRIMAUTH CORRNAME CONNTYPE
ORIGAUTH CORRNMBR INSTANCE     EVENT TIMESTAMP            --- L O C K   R E S O U R C E ---
PLANNAME CONNECT               RELATED TIMESTAMP EVENT    TYPE      NAME                    EVENT SPECIFIC DATA
------------------------------ ----------------- -------- --------- ----------------------- ----------------------------------------
FPB      FPBPARAL TSO          15:25:27.23692350 TIMEOUT  PARTITION DB  =PARADABA           REQUEST =LOCK       UNCONDITIONAL
FPB      'BLANK'  AB09C533F92E N/P                                  OB  =TAB1TS             STATE   =S          ZPARM INTERVAL=  300
PARALLEL BATCH                                                      PART= 1                 DURATION=COMMIT     INTERV.COUNTER=    1
                                                                                            HASH    =X'000020E0'
                                                                                            ------------ HOLDERS/WAITERS -----------
                                                                                            HOLDER
                                                                                            LUW='BLANK'.IPSAQ421.AB09C51F32CB
                                                                                            MEMBER  =N/P        CONNECT =TSO
                                                                                            PLANNAME=DSNESPRR   CORRID=EOA
                                                                                            DURATION=COMMIT     PRIMAUTH=KARELLE
                                                                                            STATE   =X

KARL     KARL     TSO          15:30:32.97267562 TIMEOUT  PARTITION DB  =PARADABA           REQUEST =LOCK       UNCONDITIONAL
KARL     'BLANK'  AB09C65528E6 N/P                                  OB  =TAB1TS             STATE   =IS         ZPARM INTERVAL=  300
PARALLEL TSO                                                        PART= 1                 DURATION=COMMIT     INTERV.COUNTER=    1
                                                                                            HASH    =X'000020E0'
                                                                                            ------------ HOLDERS/WAITERS -----------
                                                                                            HOLDER
                                                                                            LUW='BLANK'.IPSAQ421.AB09C51F32CB
                                                                                            MEMBER  =N/P        CONNECT =TSO
                                                                                            PLANNAME=DSNESPRR   CORRID=EOA
                                                                                            DURATION=COMMIT     PRIMAUTH=DAVE
                                                                                            STATE   =X
                                                                                            ENDUSER =DAVEUSER
                                                                                            WSNAME  =DAVEWS
                                                                                            TRANS   =DAVES TRANSACTION
LOCKING TRACE COMPLETE
```

Figure 89. Portion of PM Locking Trace - Lockout

At this point in the investigation, the following information is known:

- The applications that contend for resources
- The page sets for which there is contention
- The impact, frequency, and type of the contentions

The application or data design must be reviewed to reduce the contention.

## Corrective decisions

The above discussion is a general approach when lock suspensions are unacceptably long or timeouts occur. In such cases, the DB2 performance trace for locking and the DB2 PM reports can be used to isolate the resource causing the suspensions. Locking Report - Lockout identifies the resources involved. Locking Trace - Lockout tells what contending process (agent) caused the timeout.

In Figure 87 on page 705, the number of suspensions is low (only 2) and both have ended in a timeout. Rather than use the DB2 performance trace for locking, use the preferred option, DB2 statistics class 3 and DB2 performance trace class 1. Then produce the DB2 PM locking timeout report to obtain the information necessary to reduce overheads.

For specific information about DB2 PM reports and their usage, see *DB2 PM for OS/390 Report Reference Volume 1*, *DB2 PM for OS/390 Report Reference Volume 2* and *DB2 PM for OS/390 Online Monitor User's Guide*.

## Deadlock detection scenarios

The following section includes an examination of two different deadlock scenarios and an explanation of the use of the DB2 PM deadlock detail report to determine the cause of the deadlock.

The DB2 PM report Locking Trace - Deadlock formats the information contained in trace record IFCID 172 (statistics class 3). The report outlines all the resources and agents involved in a deadlock and the significant locking parameters, such as lock state and duration, related to their requests.

These examples assume that statistics class 3 and performance class 1 are activated. Performance class 1 is activated to get IFCID 105 records, which contain the translated names for the database ID and the page set OBID.

The scenarios that follow use three of the DB2 sample tables, DEPT, PROJ, and ACT. They are all defined with LOCKSIZE ANY. Type 2 indexes are used to access all three tables. As a result, contention for locks is only on data pages.

### Scenario 1: Two-way deadlock, two resources

In this classic deadlock example, two agents contend for resources; the result is a deadlock in which one of the agents is rolled back. Two transactions and two resources are involved.

First, transaction LOC2A acquires a lock on one resource while transaction LOC2B acquires a lock on another. Next, the two transactions each request locks on the resource held by the other.

The transactions execute as follows:

**LOC2A**

1. Declare and open a cursor for update on DEPT and fetch from page 2.
2. Declare and open a cursor for update on PROJ and fetch from page 8.
3. Update page 2.
4. Update page 8.
5. Close both cursors and commit.

**LOC2B**

1. Declare and open a cursor for update on PROJ and fetch from page 8.
2. Declare and open a cursor for update on DEPT and fetch from page 2.
3. Update page 8.
4. Update page 2.
5. Close both cursors and commit.

Events take place in the following sequence:

1. LOC2A obtains a U lock on page 2 in table DEPT, to open its cursor for update.

2. LOC2B obtains a U lock on a page 8 in table PROJ, to open its cursor for update.
3. LOC2A attempts to access page 8, to open its cursor but cannot proceed because of the lock held by LOC2B.
4. LOC2B attempts to access page 2, to open its cursor but cannot proceed because of the lock held by LOC2A.

DB2 selects one of the transactions and rolls it back, releasing its locks. That allows the other transaction to proceed to completion and release its locks also.

Figure 90 shows the DB2 PM Locking Trace - Deadlock report produced for this situation.

The report shows that the only transactions involved came from plans LOC2A and LOC2B. Both transactions came in from BATCH.

```
⋮

PRIMAUTH CORRNAME CONNTYPE
ORIGAUTH CORRNMBR INSTANCE     EVENT TIMESTAMP           --- L O C K   R E S O U R C E ---
PLANNAME CONNECT               RELATED TIMESTAMP EVENT    TYPE      NAME                    EVENT SPECIFIC DATA
------------------------------ ----------------- -------- --------- ----------------------- ----------------------------------------
SYSADM   RUNLOC2A TSO          20:32:30.68850025 DEADLOCK                                   COUNTER =    2      WAITERS =    2
SYSADM   'BLANK'  AADD32FD8A8C N/P                                                        TSTAMP  =04/02/95 20:32:30.68
LOC2A    BATCH                                            DATAPAGE  DB  =DSN8D42A           HASH    =X'01060304'
    A                                                               OB  =DEPT               ---------------- BLOCKER IS HOLDER -----
                                                                    PAGE=X'000002'          LUW='BLANK'.EGTVLU2.AADD32FD8A8C
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC2A      CORRID=RUNLOC2A
                                                                                            DURATION=MANUAL     PRIMAUTH=SYSADM
                                                                                            STATE   =U
                                                                                            ---------------- WAITER ----------------
                                                                                            LUW='BLANK'.EGTVLU2.AA65FEDC1022
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC2B      CORRID=RUNLOC2B
                                                                                            DURATION=MANUAL     PRIMAUTH=KATHY
                                                                                            REQUEST =LOCK       WORTH   =    18
                                                                                            STATE   =U

                                                          DATAPAGE  DB  =DSN8D42A           HASH    =X'01060312'
                                                                    OB  =PROJ               ---------------- BLOCKER IS HOLDER -----
                                                                    PAGE=X'000008'          LUW='BLANK'.EGTVLU2.AA65FEDC1022
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC2B      CORRID=RUNLOC2B
                                                                                            DURATION=MANUAL     PRIMAUTH=KATHY
                                                                                            STATE   =U
                                                                                            ---------------- WAITER -------*VICTIM*-
                                                                                            LUW='BLANK'.EGTVLU2.AADD32FD8A8C
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC2A      CORRID=RUNLOC2A
                                                                                            DURATION=MANUAL     PRIMAUTH=SYSADM
                                                                                            REQUEST =LOCK       WORTH   =   17
                                                                                            STATE   =U
```

*Figure 90. Deadlock scenario 1: Two transactions and two resources*

The lock held by transaction 1 (LOC2A) is a data page lock on the DEPT table and is held in U state. (The value of MANUAL for duration means that, if the plan was bound with isolation level CS and the page was not updated, then DB2 is free to release the lock before the next commit point.)

Transaction 2 (LOC2B) was requesting a lock on the same resource, also of mode U and hence incompatible.

The specifications of the lock held by transaction 2 (LOC2B) are the same. Transaction 1 was requesting an incompatible lock on the same resource. Hence, the deadlock.

Finally, note that the entry in the trace, identified at **A**, is LOC2A. That is the selected thread (the “victim”) whose work is rolled back to let the other proceed.

## Scenario 2: Three-way deadlock, three resources

In this scenario, three agents contend for resources and the result is a deadlock in which one of the agents is rolled back. Three transactions and three resources are involved.

First, the three transactions each acquire a lock on a different resource. LOC3A then requests a lock on the resource held by LOC3B, LOC3B requests a lock on the resource held by LOC3C, and LOC3C requests a lock on the resource held by LOC3A.

The transactions execute as follows:

**LOC3A**

1. Declare and open a cursor for update on DEPT and fetch from page 2.
2. Declare and open a cursor for update on PROJ and fetch from page 8.
3. Update page 2.
4. Update page 8.
5. Close both cursors and commit.

**LOC3B**

1. Declare and open a cursor for update on PROJ and fetch from page 8.
2. Declare and open a cursor for update on ACT and fetch from page 6.
3. Update page 6.
4. Update page 8.
5. Close both cursors and commit.

**LOC3C**

1. Declare and open a cursor for update on ACT and fetch from page 6.
2. Declare and open a cursor for update on DEPT and fetch from page 2.
3. Update page 6.
4. Update page 2.
5. Close both cursors and commit.

Events take place in the following sequence:

1. LOC3A obtains a U lock on page 2 in DEPT, to open its cursor for update.
2. LOC3B obtains a U lock on page 8 in PROJ, to open its cursor for update.
3. LOC3C obtains a U lock on page 6 in ACT, to open its cursor for update.
4. LOC3A attempts to access page 8 in PROJ but cannot proceed because of the lock held by LOC3B.
5. LOC3B attempts to access page 6 in ACT cannot proceed because of the lock held by LOC3C.
6. LOC3C attempts to access page 2 in DEPT but cannot proceed, because of the lock held by LOC3A.

DB2 rolls back LOC3C and releases its locks. That allows LOC3B to complete and release the lock on PROJ so that LOC3A can complete. LOC3C can then retry.

Figure 91 shows the DB2 PM Locking Trace - Deadlock report produced for this situation.

⋮

```
PRIMAUTH CORRNAME CONNTYPE
ORIGAUTH CORRNMBR INSTANCE     EVENT TIMESTAMP           --- L O C K   R E S O U R C E ---
PLANNAME CONNECT               RELATED TIMESTAMP EVENT    TYPE      NAME                    EVENT SPECIFIC DATA
------------------------------ ----------------- -------- --------- ----------------------- ----------------------------------------
SYSADM   RUNLOC3C TSO          15:10:39.33061694 DEADLOCK                                   COUNTER =    3      WAITERS =    3
SYSADM   'BLANK'  AADE2CF16F34 N/P                                                          TSTAMP  =04/03/95 15:10:39.31
LOC3C    BATCH                                            DATAPAGE  DB  =DSN8D42A           HASH    =X'01060312'
                                                                    OB  =PROJ               ---------------- BLOCKER IS HOLDER------
                                                                    PAGE=X'000008'          LUW='BLANK'.EGTVLU2.AAD15D373533
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC3B      CORRID=RUNLOC3B
                                                                                            DURATION=MANUAL     PRIMAUTH=JULIE
                                                                                            STATE   =U
                                                                                            ---------------- WAITER ----------------
                                                                                            LUW='BLANK'.EGTVLU2.AB33745CE357
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC3A      CORRID=RUNLOC3A
                                                                                            DURATION=MANUAL     PRIMAUTH=BOB
                                                                                            REQUEST =LOCK       WORTH   =   18
                                                                                            STATE   =U
                                                                                            ---------- BLOCKER IS HOLDER --*VICTIM*-
                                                                                            LUW='BLANK'.EGTVLU2.AAD15D373533
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC3C      CORRID  =RUNLOC3C
                                                                                            DURATION=MANUAL     PRIMAUTH=SYSADM
                                                                                            STATE   =U
                                                                                            ---------------- WAITER ----------------
                                                                                            LUW='BLANK'.EGTVLU2.AB33745CE357
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC3B      CORRID  =RUNLOC3B
                                                                                            DURATION=MANUAL     PRIMAUTH=JULIE
                                                                                            REQUEST =LOCK       WORTH   =   18
                                                                                            STATE   =U
                                                                                            ---------- BLOCKER IS HOLDER -----------
                                                                                            LUW='BLANK'.EGTVLU2.AAD15D373533
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC3A      CORRID  =RUNLOC3A
                                                                                            DURATION=MANUAL     PRIMAUTH=BOB
                                                                                            STATE   =U
                                                                                            ---------------- WAITER -------*VICTIM*-
                                                                                            LUW='BLANK'.EGTVLU2.AB33745CE357
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC3C      CORRID  =RUNLOC3C
                                                                                            DURATION=MANUAL     PRIMAUTH=SYSADM
                                                                                            REQUEST =LOCK       WORTH   =   18
                                                                                            STATE   =U
```

*Figure 91. Deadlock scenario 2: Three transactions and three resources*

# Chapter 31. Tuning your queries

The information under this heading, up to the end of this chapter, is Product-sensitive Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

This chapter tells you how to improve the performance of your queries. It begins with:

- “General tips and questions”

For more detailed information and suggestions, see:

- “Writing efficient predicates” on page 714
- “Using host variables efficiently” on page 734
- “Writing efficient subqueries” on page 738
- “Using scrollable cursors efficiently” on page 744
- “Writing efficient queries on views with UNION operators” on page 745

If you still have performance problems after you have tried the suggestions in these sections, there are other, more risky techniques you can use. See “Special techniques to influence access path selection” on page 746 for information.

## General tips and questions

***Recommendation:*** If you have a query that is performing poorly, first go over the following checklist to see that you have not overlooked some of the basics.

### Is the query coded as simply as possible?

Make sure the SQL query is coded as simply and efficiently as possible. Make sure that no unused columns are selected and that there is no unneeded ORDER BY or GROUP BY.

### Are all predicates coded correctly?

***Indexable predicates:*** Make sure all the predicates that you think should be indexable are coded so that they can be indexable. Refer to Table 101 on page 719 to see which predicates are indexable and which are not.

***Unintentionally redundant or unnecessary predicates:*** Try to remove any predicates that are unintentionally redundant or not needed; they can slow down performance.

***Declared lengths of host variables:*** Make sure that the declared length of any host variable is no greater than the length attribute of the data column it is compared to. If the declared length is greater, the predicate is stage 2 and cannot be a matching predicate for an index scan.

For example, assume that a host variable and an SQL column are defined as follows:

| **Assembler declaration** | **SQL definition** |
|---|---|
| MYHOSTV DS PLn ’value’ | COL1 DECIMAL(6,3) |

When ’n’ is used, the precision of the host variable is ’2n-1’. If n = 4 and value = ’123.123’, then a predicate such as `WHERE COL1 = :MYHOSTV` is not a matching predicate for an index scan because the precisions are different. One way to avoid an inefficient predicate using decimal host variables is to declare the host variable without the ’Ln’ option:

```
    MYHOSTV  DS P'123.123'
```

This guarantees the same host variable declaration as the SQL column definition.

## Are there subqueries in your query?

If your query uses subqueries, see “Writing efficient subqueries” on page 738 to understand how DB2 executes subqueries. There are no absolute rules to follow when deciding how or whether to code a subquery. But these are general guidelines:

- If there are efficient indexes available on the tables in the subquery, then a correlated subquery is likely to be the most efficient kind of subquery.
- If there are no efficient indexes available on the tables in the subquery, then a noncorrelated subquery would likely perform better.
- If there are multiple subqueries in any parent query, make sure that the subqueries are ordered in the most efficient manner.

Consider the following illustration. Assume that there are 1000 rows in MAIN_TABLE.

```
SELECT * FROM MAIN_TABLE
WHERE TYPE IN (subquery 1)
  AND
  PARTS IN (subquery 2);
```

Assuming that subquery 1 and subquery 2 are the same type of subquery (either correlated or noncorrelated), DB2 evaluates the subquery predicates in the order they appear in the WHERE clause. Subquery 1 rejects 10% of the total rows, and subquery 2 rejects 80% of the total rows.

The predicate in subquery 1 (which is referred to as P1) is evaluated 1,000 times, and the predicate in subquery 2 (which is referred to as P2) is evaluated 900 times, for a total of 1,900 predicate checks. However, if the order of the subquery predicates is reversed, P2 is evaluated 1000 times, but P1 is evaluated only 200 times, for a total of 1,200 predicate checks.

It appears that coding P2 before P1 would be more efficient if P1 and P2 take an equal amount of time to execute. However, if P1 is 100 times faster to evaluate than P2, then it might be advisable to code subquery 1 first. If you notice a performance degradation, consider reordering the subqueries and monitoring the results. Consult “Writing efficient subqueries” on page 738 to help you understand what factors make one subquery run more slowly than another.

If you are in doubt, run EXPLAIN on the query with both a correlated and a noncorrelated subquery. By examining the EXPLAIN output and understanding your data distribution and SQL statements, you should be able to determine which form is more efficient.

This general principle can apply to all types of predicates. However, because subquery predicates can potentially be thousands of times more processor- and I/O-intensive than all other predicates, it is most important to make sure they are coded in the correct order.

DB2 always performs all noncorrelated subquery predicates before correlated subquery predicates, regardless of coding order.

Refer to “DB2 predicate manipulation” on page 728 to see in what order DB2 will evaluate predicates and when you can control the evaluation order.

## Does your query involve column functions?

If your query involves column functions, make sure that they are coded as simply as possible; this increases the chances that they will be evaluated when the data is retrieved, rather than afterward. In general, a column function performs best when evaluated during data access and next best when evaluated during DB2 sort. Least preferable is to have a column function evaluated after the data has been retrieved. Refer to “When are column functions evaluated? (COLUMN_FN_EVAL)” on page 805 for help in using EXPLAIN to get the information you need.

For column functions to be evaluated during data retrieval, the following conditions must be met for all column functions in the query:

- There must be no sort needed for GROUP BY. Check this in the EXPLAIN output.
- There must be no stage 2 (residual) predicates. Check this in your application.
- There must be no distinct set functions such as COUNT(DISTINCT C1).
- If the query is a join, all set functions must be on the last table joined. Check this by looking at the EXPLAIN output.
- All column functions must be on single columns with no arithmetic expressions.
- The column function is *not* one of the following column functions:
- STDDEV
- STDDEV_SAMP
- VAR
- VAR_SAMP

If your query involves the functions MAX or MIN, refer to “One-fetch access (ACCESSTYPE=I1)” on page 811 to see whether your query could take advantage of that method.

## Do you have an input variable in the predicate of a static SQL query?

When host variables or parameter markers are used in a query, the actual values are not known when you bind the package or plan that contains the query. DB2 therefore uses a default filter factor to determine the best access path for an SQL statement. If that access path proves to be inefficient, there are several things you can do to obtain a better access path.

See “Using host variables efficiently” on page 734 for more information.

## Do you have a problem with column correlation?

Two columns in a table are said to be correlated if the values in the columns do not vary independently.

DB2 might not determine the best access path when your queries include correlated columns. If you think you have a problem with column correlation, see “Column correlation” on page 731 for ideas on what to do about it.

## Can your query be written to use a noncolumn expression?

The following predicate combines a column, SALARY, with values that are not from columns on one side of the operator:

```
WHERE SALARY + (:hv1 * SALARY) > 50000
```

If you rewrite the predicate in the following way, DB2 can evaluate it more efficiently:

```
WHERE SALARY > 50000/(1 + :hv1)
```

In the second form, the column is by itself on one side of the operator, and all the other values are on the other side of the operator. The expression on the right is called a *noncolumn expression*. DB2 can evaluate many predicates with noncolumn expressions at an earlier stage of processing called *stage 1*, so the queries take less time to run.

For more information on noncolumn expressions and stage 1 processing, see “Properties of predicates”.

## Writing efficient predicates

***Definition:*** *Predicates* are found in the clauses WHERE, HAVING or ON of SQL statements; they describe attributes of data. They are usually based on the columns of a table and either qualify rows (through an index) or reject rows (returned by a scan) when the table is accessed. The resulting qualified or rejected rows are independent of the access path chosen for that table.

***Example:*** The query below has three predicates: an equal predicate on C1, a BETWEEN predicate on C2, and a LIKE predicate on C3.

```
SELECT * FROM T1
   WHERE C1 = 10 AND
         C2 BETWEEN 10 AND 20 AND
         C3 NOT LIKE 'A%'
```

***Effect on access paths:*** This section explains the effect of predicates on access paths. Because SQL allows you to express the same query in different ways, knowing how predicates affect path selection helps you write queries that access data efficiently.

This section describes:
- “Properties of predicates”
- “General rules about predicate evaluation” on page 717
- “Predicate filter factors” on page 723
- “DB2 predicate manipulation” on page 728
- “Column correlation” on page 731

## Properties of predicates

Predicates in a HAVING clause are not used when selecting access paths; hence, in this section the term ’predicate’ means a predicate after WHERE or ON.

A predicate influences the selection of an access path because of:
- Its **type**, as described in “Predicate types” on page 715
- Whether it is **indexable**, as described in “Indexable and nonindexable predicates” on page 716
- Whether it is **stage 1** or **stage 2**

- Whether it contains a ROWID column, as described in “Is direct row access possible? (PRIMARY_ACCESSTYPE = D)” on page 801

There are special considerations for “Predicates in the ON clause” on page 717.

***Definitions:*** Predicates are identified as:

**Simple or compound**
A *compound* predicate is the result of two predicates, whether simple or compound, connected together by AND or OR Boolean operators. All others are *simple*.

**Local or join**
*Local predicates* reference only one table. They are local to the table and restrict the number of rows returned for that table. *Join predicates* involve more than one table or correlated reference. They determine the way rows are joined from two or more tables. For examples of their use, see “Interpreting access to two or more tables (join)” on page 812.

**Boolean term**
Any predicate that is not contained by a compound OR predicate structure is a *Boolean term*. If a Boolean term is evaluated false for a particular row, the whole WHERE clause is evaluated false for that row.

## Predicate types

The type of a predicate depends on its operator or syntax, as listed below. The type determines what type of processing and filtering occurs when the predicate is evaluated.

**Type** **Definition**

**Subquery**
Any predicate that includes another SELECT statement. Example: C1 IN (SELECT C10 FROM TABLE1)

**Equal** Any predicate that is not a subquery predicate and has an equal operator and no NOT operator. Also included are predicates of the form C1 IS NULL. Example: C1=100

**Range**
Any predicate that is not a subquery predicate and has an operator in the following list: >, >=, <, <=, LIKE, or BETWEEN. Example: C1>100

**IN-list** A predicate of the form column IN (list of values). Example: C1 IN (5,10,15)

**NOT** Any predicate that is not a subquery predicate and contains a NOT operator. Example: COL1 <> 5 or COL1 NOT BETWEEN 10 AND 20.

***Example: Influence of type on access paths:*** The following two examples show how the predicate type can influence DB2’s choice of an access path. In each one, assume that a unique index I1 (C1) exists on table T1 (C1, C2), and that all values of C1 are positive integers.

The query,

```
SELECT C1, C2 FROM T1 WHERE C1 >= 0;
```

has a range predicate. However, the predicate does not eliminate any rows of T1. Therefore, it could be determined during bind that a table space scan is more efficient than the index scan.

The query,

```
SELECT * FROM T1 WHERE C1 = 0;
```

has an equal predicate. DB2 chooses the index access in this case, because only one scan is needed to return the result.

### Indexable and nonindexable predicates

***Definition:*** *Indexable* predicate types can match index entries; other types cannot. Indexable predicates might not become matching predicates of an index; it depends on the indexes that are available and the access path chosen at bind time.

***Examples:*** If the employee table has an index on the column LASTNAME, the following predicate can be a matching predicate:

```
SELECT * FROM DSN8710.EMP WHERE LASTNAME = 'SMITH';
```

The following predicate cannot be a matching predicate, because it is not indexable.

```
SELECT * FROM DSN8710.EMP WHERE SEX <> 'F';
```

***Recommendation:*** To make your queries as efficient as possible, use indexable predicates in your queries and create suitable indexes on your tables. Indexable predicates allow the possible use of a matching index scan, which is often a very efficient access path.

### Stage 1 and stage 2 predicates

***Definition:*** Rows retrieved for a query go through two stages of processing.

1. *Stage 1* predicates (sometimes called *sargable*) can be applied at the first stage.
2. *Stage 2* predicates (sometimes called *nonsargable* or *residual*) cannot be applied until the second stage.

The following items determine whether a predicate is stage 1:

- Predicate syntax

  See Table 101 on page 719 for a list of simple predicates and their types. See Examples of predicate properties for information on compound predicate types.
- Type and length of constants in the predicate

  A simple predicate whose syntax classifies it as stage 1 might not be stage 1 because it contains constants and columns whose types or lengths disagree. For example, the following predicates are not stage 1:
  - CHARCOL='ABCDEFG', where CHARCOL is defined as CHAR(6)
  - SINTCOL>34.5, where SINTCOL is defined as SMALLINT

  The first predicate is not stage 1 because the length of the column is shorter than the length of the constant. The second predicate is not stage 1 because the data types of the column and constant are not the same.
- Whether DB2 evaluates the predicate before or after a join operation. A predicate that is evaluated after a join operation is always a stage 2 predicate.

***Examples:*** All indexable predicates are stage 1. The predicate C1 LIKE %BC is also stage 1, but is not indexable.

***Recommendation:*** Use stage 1 predicates whenever possible.

### Boolean term (BT) predicates

***Definition:*** A *Boolean term predicate*, or *BT predicate*, is a simple or compound predicate that, when it is evaluated false for a particular row, makes the entire WHERE clause false for that particular row.

***Examples:*** In the following query P1, P2 and P3 are simple predicates:

```
SELECT * FROM T1 WHERE P1 AND (P2 OR P3);
```

- P1 is a simple BT predicate.
- P2 and P3 are simple non-BT predicates.
- P2 OR P3 is a compound BT predicate.
- P1 AND (P2 OR P3) is a compound BT predicate.

***Effect on access paths:*** In single index processing, only Boolean term predicates are chosen for matching predicates. Hence, only indexable Boolean term predicates are candidates for matching index scans. To match index columns by predicates that are not Boolean terms, DB2 considers multiple index access.

In join operations, Boolean term predicates can reject rows at an earlier stage than can non-Boolean term predicates.

***Recommendation:*** For join operations, choose Boolean term predicates over non-Boolean term predicates whenever possible.

## Predicates in the ON clause

The ON clause supplies the join condition in an outer join. For a full outer join, the clause can use only equal predicates. For other outer joins, the clause can use any predicates except predicates that contain subqueries.

For left and right outer joins, and for inner joins, join predicates in the ON clause are treated the same as other stage 1 and stage 2 predicates. A stage 2 predicate in the ON clause is treated as a stage 2 predicate of the inner table.

For full outer join, the ON clause is evaluated during the join operation like a stage 2 predicate.

In an outer join, predicates that are evaluated after the join are stage 2 predicates. Predicates in a table expression can be evaluated before the join and can therefore be stage 1 predicates.

For example, in the following statement,

```
SELECT * FROM (SELECT * FROM DSN8710.EMP
   WHERE EDLEVEL  > 100) AS X FULL JOIN DSN8710.DEPT
      ON X.WORKDEPT  = DSN8710.DEPT.DEPTNO;
```

the predicate “EDLEVEL > 100” is evaluated before the full join and is a stage 1 predicate. For more information on join methods, see “Interpreting access to two or more tables (join)” on page 812.

## General rules about predicate evaluation

***Recommendations:***

1. In terms of resource usage, the earlier a predicate is evaluated, the better.
2. Stage 1 predicates are better than stage 2 predicates because they qualify rows earlier and reduce the amount of processing needed at stage 2.
3. When possible, try to write queries that evaluate the most restrictive predicates first. When predicates with a high filter factor are processed first, unnecessary rows are screened as early as possible, which can reduce processing cost at a later stage. However, a predicate’s restrictiveness is only effective among predicates of the same type and the same evaluation stage. For information about filter factors, see “Predicate filter factors” on page 723.

## Order of evaluating predicates

Two sets of rules determine the order of predicate evaluation.

The first set:

1. Indexable predicates are applied first. All matching predicates on index key columns are applied first and evaluated when the index is accessed.

   First, stage 1 predicates that have not been picked as matching predicates but still refer to index columns are applied to the index. This is called *index screening*.
2. Other stage 1 predicates are applied next.

   After data page access, stage 1 predicates are applied to the data.
3. Finally, the stage 2 predicates are applied on the returned data rows.

The second set of rules describes the order of predicate evaluation within each of the above stages:

1. All equal predicates a (including column IN *list*, where *list* has only one element).
2. All range predicates and predicates of the form *column* IS NOT NULL
3. All other predicate types are evaluated.

After both sets of rules are applied, predicates are evaluated in the order in which they appear in the query. Because you specify that order, you have some control over the order of evaluation.

## Summary of predicate processing

Table 101 on page 719 lists many of the simple predicates and tells whether those predicates are indexable or stage 1. The following terms are used:

- *non subq* means a noncorrelated subquery.
- *cor subq* means a correlated subquery.
- *op* is any of the operators >, >=, <, <=, ¬>, ¬<.
- *value* is a constant, host variable, or special register.
- *pattern* is any character string that does *not* start with the special characters for percent (%) or underscore (_).
- *char* is any character string that does *not* include the special characters for percent (%) or underscore (_).
- *expression* is any expression that contains arithmetic operators, scalar functions, column functions, concatenation operators, columns, constants, host variables, special registers, or date or time expressions.
- *noncol expr* is a noncolumn expression, which is any expression that does not contain a column. That expression can contain arithmetic operators, scalar functions, concatenation operators, constants, host variables, special registers, or date or time expressions.

  An example of a noncolumn expression is

  ```
  CURRENT DATE - 50 DAYS
  ```
- *predicate* is a predicate of any type.

In general, if you form a compound predicate by combining several simple predicates with OR operators, the result of the operation has the same characteristics as the simple predicate that is evaluated latest. For example, if two indexable predicates are combined with an OR operator, the result is indexable. If a stage 1 predicate and a stage 2 predicate are combined with an OR operator, the result is stage 2.

*Table 101. Predicate types and processing*

| Predicate Type | Index-able? | Stage 1? | Notes |
|---|---|---|---|
| COL = *value* | Y | Y | 13 |
| COL = *noncol expr* | Y | Y | 9, 11, 12 |
| COL IS NULL | Y | Y | |
| COL *op value* | Y | Y | |
| COL *op noncol expr* | Y | Y | 9, 11 |
| COL BETWEEN *value1* AND *value2* | Y | Y | |
| COL BETWEEN *noncol expr1* AND *noncol expr2* | Y | Y | 9, 11 |
| *value* BETWEEN COL1 AND COL2 | N | N | |
| COL BETWEEN COL1 AND COL2 | N | N | 10 |
| COL BETWEEN *expression1* AND *expression2* | N | N | 7 |
| COL LIKE '*pattern*' | Y | Y | 6 |
| COL IN (*list*) | Y | Y | 14 |
| COL <> *value* | N | Y | 8 |
| COL <> *noncol expr* | N | Y | 8, 11 |
| COL IS NOT NULL | N | Y | |
| COL NOT BETWEEN *value1* AND *value2* | N | Y | |
| COL NOT BETWEEN *noncol expr1* AND *noncol expr2* | N | Y | 11 |
| *value* NOT BETWEEN COL1 AND COL2 | N | N | |
| COL NOT IN (*list*) | N | Y | |
| COL NOT LIKE ' *char*' | N | Y | 6 |
| COL LIKE '%*char*' | N | Y | 1, 6 |
| COL LIKE '_*char*' | N | Y | 1, 6 |
| COL LIKE *host variable* | Y | Y | 2, 6 |
| T1.COL = T2.COL | Y | Y | 16 |
| T1.COL *op* T2.COL | Y | Y | 3 |
| T1.COL <> T2.COL | N | Y | 3 |
| T1.COL1 = T1.COL2 | N | N | 4 |
| T1.COL1 *op* T1.COL2 | N | N | 4 |
| T1.COL1 <> T1.COL2 | N | N | 4 |
| COL=(*non subq*) | Y | Y | 15 |
| COL = ANY (*non subq*) | N | N | |

*Table 101. Predicate types and processing (continued)*

| **Predicate Type** | **Index-able?** | **Stage 1?** | **Notes** |
|---|---|---|---|
| COL = ALL (*non subq*) | N | N | |
| COL *op* (*non subq*) | Y | Y | 15 |
| COL *op* ANY (*non subq*) | Y | Y | |
| COL *op* ALL (*non subq*) | Y | Y | |
| COL <> (*non subq*) | N | Y | |
| COL <> ANY (*non subq*) | N | N | |
| COL <> ALL (*non subq*) | N | N | |
| COL IN (*non subq*) | Y | Y | |
| (COL1,...COL*n*) IN (*non subq*) | Y | Y | |
| COL NOT IN (*non subq*) | N | N | |
| (COL1,...COL*n*) NOT IN (*non subq*) | N | N | |
| COL = (*cor subq*) | N | N | 5 |
| COL = ANY (*cor subq*) | N | N | |
| COL = ALL (*cor subq*) | N | N | |
| COL *op* (*cor subq*) | N | N | 5 |
| COL *op* ANY (*cor subq*) | N | N | |
| COL *op* ALL (*cor subq*) | N | N | |
| COL <> (*cor subq*) | N | N | 5 |
| COL <> ANY (*cor subq*) | N | N | |
| COL <> ALL (*cor subq*) | N | N | |
| COL IN (*cor subq*) | N | N | |
| (COL1,...COL*n*) IN (*cor subq*) | N | N | |
| COL NOT IN (*cor subq*) | N | N | |
| (COL1,...COL*n*) NOT IN (*cor subq*) | N | N | |
| EXISTS (*subq*) | N | N | |
| NOT EXISTS (*subq*) | N | N | |
| COL = *expression* | Y | Y | 7 |
| *expression = value* | N | N | |
| *expression* <> *value* | N | N | |
| *expression op value* | N | N | |
| *expression op* (subquery) | N | N | |

**Notes to Table 101 on page 719:**

1. Indexable only if an ESCAPE character is specified and used in the LIKE predicate. For example, COL LIKE '+%*char*' ESCAPE '+' is indexable.
2. Indexable only if the pattern in the host variable is an indexable constant (for example, host variable='*char%*').
3. Within each statement, the columns are of the same type. Examples of different column types include:
   - Different data types, such as INTEGER and DECIMAL

- Different numeric column lengths, such as DECIMAL(5,0) and DECIMAL(15,0)
- Different decimal scales, such as DECIMAL(7,3) and DECIMAL(7,4).

  The following columns are considered to be of the same types:

  - Columns of the same data type but different subtypes.
  - Columns of the same data type, but different nullability attributes. (For example, one column accepts nulls but the other does not.)

4. If both COL1 and COL2 are from the same table, access through an index on either one is not considered for these predicates. However, the following query is an exception:

   ```
   SELECT *  FROM T1 A, T1 B WHERE A.C1 = B.C2;
   ```

   By using correlation names, the query treats one table as if it were two separate tables. Therefore, indexes on columns C1 and C2 are considered for access.
5. If the subquery has already been evaluated for a given correlation value, then the subquery might not have to be reevaluated.
6. Not indexable or stage 1 if a field procedure exists on that column.
7. Under any of the following circumstances, the predicate is stage 1 and indexable:
   - COL is of type INTEGER or SMALLINT, and *expression* is of the form:

     ```
     integer-constant1 arithmetic-operator integer-constant2
     ```
   - COL is of type DATE, TIME, or TIMESTAMP, and:
     - *expression* is of any of these forms:

       ```
       datetime-scalar-function(character-constant)
       datetime-scalar-function(character-constant) + labeled-duration
       datetime-scalar-function(character-constant) - labeled-duration
       ```
     - The type of *datetime-scalar-function*(*character-constant*) matches the type of COL.
     - The numeric part of *labeled-duration* is an integer.
     - *character-constant* is:
       - Greater than 7 characters long for the DATE scalar function; for example, '1995-11-30'.
       - Greater than 14 characters long for the TIMESTAMP scalar function; for example, '1995-11-30-08.00.00'.
       - Any length for the TIME scalar function.
8. The processing for WHERE NOT COL = *value* is like that for WHERE COL <> *value*, and so on.
9. If *noncol expr*, *noncol expr1*, or *noncol expr2* is a noncolumn expression of one of these forms, then the predicate is not indexable:
   - *noncol expr* + 0
   - *noncol expr* - 0
   - *noncol expr* * 1
   - *noncol expr* / 1
   - *noncol expr* CONCAT *empty string*
10. COL, COL1, and COL2 can be the same column or different columns. The columns can be in the same table or different tables.
11. To ensure that the predicate is indexable and stage 1, make the data type and length of the column and the data type and length of the result of the noncolumn expression the same. For example, if the predicate is:

```
COL op scalar function
```

and the scalar function is HEX, SUBSTR, DIGITS, CHAR, or CONCAT, then the type and length of the result of the scalar function and the type and length of the column must be the same for the predicate to be indexable and stage 1.

12. Under these circumstances, the predicate is stage 2:
    - *noncol expr* is a case expression.
    - *non col expr* is the product or the quotient of two noncolumn expressions, that product or quotient is an integer value, and COL is a FLOAT or a DECIMAL column.
13. If COL has the ROWID data type, DB2 tries to use direct row access instead of index access or a table space scan.
14. If COL has the ROWID data type, and an index is defined on COL, DB2 tries to use direct row access instead of index access.
15. Not indexable and not stage 1 if COL is not null and the noncorrelated subquery SELECT clause entry can be null.
16. If the columns are numeric columns, they must have the same data type, length, and precision to be stage 1 and indexable. For character columns, the columns can be of different types and lengths. For example, predicates with the following column types and lengths are stage 1 and indexable:
    - CHAR(5) and CHAR(20)
    - VARCHAR(5) and CHAR(5)
    - VARCHAR(5) and CHAR(20)

## Examples of predicate properties

Assume that predicate P1 and P2 are simple, stage 1, indexable predicates:

P1 AND P2 is a compound, stage 1, indexable predicate.
P1 OR P2 is a compound, stage 1 predicate, not indexable except by a union of RID lists from two indexes.

The following examples of predicates illustrate the general rules shown in Table 101 on page 719. In each case, assume that there is an index on columns (C1,C2,C3,C4) of the table and that 0 is the lowest value in each column.

- WHERE C1=5 AND C2=7

  Both predicates are stage 1 and the compound predicate is indexable. A matching index scan could be used with C1 and C2 as matching columns.
- WHERE C1=5 AND C2>7

  Both predicates are stage 1 and the compound predicate is indexable. A matching index scan could be used with C1 and C2 as matching columns.
- WHERE C1>5 AND C2=7

  Both predicates are stage 1, but only the first matches the index. A matching index scan could be used with C1 as a matching column.
- WHERE C1=5 OR C2=7

  Both predicates are stage 1 but not Boolean terms. The compound is indexable. When DB2 considers multiple index access for the compound predicate, C1 and C2 can be matching columns. For single index access, C1 and C2 can be only index screening columns.
- WHERE C1=5 OR C2<>7

  The first predicate is indexable and stage 1, and the second predicate is stage 1 but not indexable. The compound predicate is stage 1 and not indexable.
- WHERE C1>5 OR C2=7

Both predicates are stage 1 but not Boolean terms. The compound is indexable. When DB2 considers multiple index access for the compound predicate, C1 and C2 can be matching columns. For single index access, C1 and C2 can be only index screening columns.

- WHERE C1 IN (subquery) AND C2=C1

  Both predicates are stage 2 and not indexable. The index is not considered for matching index access, and both predicates are evaluated at stage 2.
- WHERE C1=5 AND C2=7 AND (C3 + 5) IN (7,8)

  The first two predicates only are stage 1 and indexable. The index is considered for matching index access, and all rows satisfying those two predicates are passed to stage 2 to evaluate the third predicate.
- WHERE C1=5 OR C2=7 OR (C3 + 5) IN (7,8)

  The third predicate is stage 2. The compound predicate is stage 2 and all three predicates are evaluated at stage 2. The simple predicates are not Boolean terms and the compound predicate is not indexable.
- WHERE C1=5 OR (C2=7 AND C3=C4)

  The third predicate is stage 2. The two compound predicates (C2=7 AND C3=C4) and (C1=5 OR (C2=7 AND C3=C4)) are stage 2. All predicates are evaluated at stage 2.
- WHERE (C1>5 OR C2=7) AND C3 = C4

  The compound predicate (C1>5 OR C2=7) is indexable and stage 1. The simple predicate C3=C4 is not stage1; so the index is not considered for matching index access. Rows that satisfy the compound predicate (C1>5 OR C2=7) are passed to stage 2 for evaluation of the predicate C3=C4.
- WHERE T1.COL1=T2.COL1 AND T1.COL2=T2.COL2

  Assume that T1.COL1 and T2.COL1 have the same data types, and T1.COL2 and T2.COL2 have the same data types. If T1.COL1 and T2.COL1 have different nullability attributes, but T1.COL2 and T2.COL2 have the same nullability attributes, and DB2 chooses a merge scan join to evaluate the compound predicate, the compound predicate is stage 1. However, if T1.COL2 and T2.COL2 also have different nullability attributes, and DB2 chooses a merge scan join, the compound predicate is not stage 1.

## Predicate filter factors

***Definition:*** The *filter factor* of a predicate is a number between 0 and 1 that estimates the proportion of rows in a table for which the predicate is true. Those rows are said to *qualify* by that predicate.

***Example:*** Suppose that DB2 can determine that column C1 of table T contains only five distinct values: A, D, Q, W and X. In the absence of other information, DB2 estimates that one-fifth of the rows have the value D in column C1. Then the predicate `C1='D'` has the filter factor 0.2 for table T.

***How DB2 uses filter factors:*** Filter factors affect the choice of access paths by estimating the number of rows qualified by a set of predicates.

For simple predicates, the filter factor is a function of three variables:

1. The literal value in the predicate; for instance, ’D’ in the previous example.
2. The operator in the predicate; for instance, ’=’ in the previous example and ’<>’ in the negation of the predicate.
3. Statistics on the column in the predicate. In the previous example, those include the information that column T.C1 contains only five values.

***Recommendation:*** You control the first two of those variables when you write a predicate. Your understanding of DB2's use of filter factors should help you write more efficient predicates.

Values of the third variable, statistics on the column, are kept in the DB2 catalog. You can update many of those values, either by running the utility RUNSTATS or by executing UPDATE for a catalog table. For information about using RUNSTATS, see "Gathering monitor and update statistics" on page 775. For information on updating the catalog manually, see "Updating catalog statistics" on page 754.

If you intend to update the catalog with statistics of your own choice, you should understand how DB2 uses:
- "Default filter factors for simple predicates"
- "Filter factors for uniform distributions"
- "Interpolation formulas" on page 725
- "Filter factors for all distributions" on page 726

## Default filter factors for simple predicates

Table 102 lists default filter factors for different types of predicates. DB2 uses those values when no other statistics exist.

***Example:*** The default filter factor for the predicate `C1 = 'D'` is 1/25 (0.04). If D is actually one of only five distinct values in column C1, the default probably does not lead to an optimal access path.

*Table 102. DB2 default filter factors by predicate type*

| **Predicate Type** | **Filter Factor** |
|---|---|
| Col = literal | 1/25 |
| Col IS NULL | 1/25 |
| Col IN (literal list) | (number of literals)/25 |
| Col *Op* literal | 1/3 |
| Col LIKE literal | 1/10 |
| Col BETWEEN literal1 and literal2 | 1/10 |

**Note:**
*Op* is one of these operators: <, <=, >, >=.
*Literal* is any constant value that is known at bind time.

## Filter factors for uniform distributions

DB2 uses the filter factors in Table 103 if:
- There is a positive value in column COLCARDF of catalog table SYSIBM.SYSCOLUMNS for the column "Col".
- There are no additional statistics for "Col" in SYSIBM.SYSCOLDIST.

***Example:*** If D is one of only five values in column C1, using RUNSTATS will put the value 5 in column COLCARDF of SYSCOLUMNS. If there are no additional statistics available, the filter factor for the predicate `C1 = 'D'` is 1/5 (0.2).

*Table 103. DB2 uniform filter factors by predicate type*

| **Predicate Type** | **Filter Factor** |
|---|---|
| Col = literal | 1/COLCARDF |
| Col IS NULL | 1/COLCARDF |
| Col IN (literal list) | number of literals /COLCARDF |

*Table 103. DB2 uniform filter factors by predicate type (continued)*

| Predicate Type | Filter Factor |
|---|---|
| Col *Op1* literal | interpolation formula |
| Col *Op2* literal | interpolation formula |
| Col LIKE literal | interpolation formula |
| Col BETWEEN literal1 and literal2 | interpolation formula |

**Note:**

*Op1* is < or <=, and the literal is not a host variable.
*Op2* is > or >=, and the literal is not a host variable.
*Literal* is any constant value that is known at bind time.

***Filter factors for other predicate types:*** The examples selected in Table 102 on page 724 and Table 103 on page 724 represent only the most common types of predicates. If P1 is a predicate and F is its filter factor, then the filter factor of the predicate NOT P1 is (1 - F). But, filter factor calculation is dependent on many things, so a specific filter factor cannot be given for all predicate types.

## Interpolation formulas

***Definition:*** For a predicate that uses a range of values, DB2 calculates the filter factor by an *interpolation formula*. The formula is based on an estimate of the ratio of the number of values in the range to the number of values in the entire column of the table.

***The formulas:*** The formulas that follow are rough estimates, subject to further modification by DB2. They apply to a predicate of the form *col op.* literal. The value of (Total Entries) in each formula is estimated from the values in columns HIGH2KEY and LOW2KEY in catalog table SYSIBM.SYSCOLUMNS for column *col*: Total Entries = (HIGH2KEY value - LOW2KEY value).

- For the operators < and <=, where the literal is not a host variable:
  (Literal value - LOW2KEY value) / (Total Entries)
- For the operators > and >=, where the literal is not a host variable:
  (HIGH2KEY value - Literal value) / (Total Entries)
- For LIKE or BETWEEN:
  (High literal value - Low literal value) / (Total Entries)

***Example:*** For column C2 in a predicate, suppose that the value of HIGH2KEY is 1400 and the value of LOW2KEY is 200. For C2, DB2 calculates (Total Entries) = 1200.

For the predicate `C1 BETWEEN 800 AND 1100`, DB2 calculates the filter factor F as:

```
F = (1100 - 800)/1200 = 1/4 = 0.25
```

***Interpolation for LIKE:*** DB2 treats a LIKE predicate as a type of BETWEEN predicate. Two values that bound the range qualified by the predicate are generated from the literal string in the predicate. Only the leading characters found before the escape character ('%' or '_') are used to generate the bounds. So if the escape character is the first character of the string, the filter factor is estimated as 1, and the predicate is estimated to reject no rows.

***Defaults for interpolation:*** DB2 might not interpolate in some cases; instead, it can use a default filter factor. Defaults for interpolation are:

- Relevant only for ranges, including LIKE and BETWEEN predicates

- Used only when interpolation is not adequate
- Based on the value of COLCARDF
- Used whether uniform or additional distribution statistics exist on the column if either of the following conditions is met:
  – The predicate does not contain constants
  – COLCARDF < 4.

Table 104 shows interpolation defaults for the operators <, <=, >, >= and for LIKE and BETWEEN.

*Table 104. Default filter factors for interpolation*

| COLCARDF | Factor for Op | Factor for LIKE or BETWEEN |
|---|---|---|
| ≥100,000,000 | 1/10,000 | 3/100,000 |
| ≥10,000,000 | 1/3,000 | 1/10,000 |
| ≥1,000,000 | 1/1,000 | 3/10,000 |
| ≥100,000 | 1/300 | 1/1,000 |
| ≥10,000 | 1/100 | 3/1,000 |
| ≥1,000 | 1/30 | 1/100 |
| ≥100 | 1/10 | 3/100 |
| ≥0 | 1/3 | 1/10 |

**Note:** Op is one of these operators: <, <=, >, >=.

## Filter factors for all distributions

RUNSTATS can generate additional statistics for a column or set of concatenated key columns of an index. DB2 can use that information to calculate filter factors. DB2 collects two kinds of distribution statistics:

**Frequency**
The percentage of rows in the table that contain a value for a column or combination of values for concatenated columns

**Cardinality**
The number of distinct values in concatenated columns

***When they are used:*** Table 105 lists the types of predicates on which these statistics are used.

*Table 105. Predicates for which distribution statistics are used*

| Type of statistic | Single column or concatenated columns | Predicates |
|---|---|---|
| Frequency | Single | COL=*literal*<br>COL IS NULL<br>COL IN (*literal-list*)<br>COL *op literal*<br>COL BETWEEN *literal* AND *literal* |
| Frequency | Concatenated | COL=*literal* |

*Table 105. Predicates for which distribution statistics are used (continued)*

| Type of statistic | Single column or concatenated columns | Predicates |
|---|---|---|
| Cardinality | Single | COL=*literal*<br>COL IS NULL<br>COL IN (*literal-list*)<br>COL *op literal*<br>COL BETWEEN *literal* AND *literal*<br>COL=*host-variable*<br>COL1=COL2 |
| Cardinality | Concatenated | COL=*literal*<br>COL=*:host-variable*<br>COL1=COL2 |

**Note:** *op* is one of these operators: <, <=, >, >=.

***How they are used:*** Columns COLVALUE and FREQUENCYF in table SYSCOLDIST contain distribution statistics. Regardless of the number of values in those columns, running RUNSTATS deletes the existing values and inserts rows for the most frequent values. If you run RUNSTATS without the FREQVAL option, RUNSTATS inserts rows for the 10 most frequent values for the first column of the specified index. If you run RUNSTATS with the FREQVAL option and its two keywords, NUMCOLS and COUNT, RUNSTATS inserts rows for concatenated columns of an index. NUMCOLS specifies the number of concatenated index columns. COUNT specifies the number of most frequent values. See Part 2 of *DB2 Utility Guide and Reference* for more information about RUNSTATS. DB2 uses the frequencies in column FREQUENCYF for predicates that use the values in column COLVALUE and assumes that the remaining data are uniformly distributed.

***Example: Filter factor for a single column***

Suppose that the predicate is `C1 IN ('3','5')` and that SYSCOLDIST contains these values for column C1:

```
COLVALUE    FREQUENCYF
  '3'         .0153
  '5'         .0859
  '8'         .0627
```

The filter factor is .0153 + .0859 = .1012.

***Example: Filter factor for correlated columns***

Suppose that columns C1 and C2 are correlated and are concatenated columns of an index. Suppose also that the predicate is `C1='3' AND C2='5'` and that SYSCOLDIST contains these values for columns C1 and C2:

```
COLVALUE    FREQUENCYF
 '1' '1'       .1176
 '2' '2'       .0588
 '3' '3'       .0588
 '3' '5'       .1176
 '4' '4'       .0588
 '5' '3'       .1764
 '5' '5'       .3529
 '6' '6'       .0588
```

The filter factor is .1176.

# DB2 predicate manipulation

In some specific cases, DB2 either modifies some predicates, or generates extra predicates. Although these modifications are transparent to you, they have a direct impact on the access path selection and your PLAN_TABLE results. This is because DB2 always uses an index access path when it is cost effective. Generating extra predicates provides more indexable predicates potentially, which creates more chances for an efficient index access path.

Therefore, to understand your PLAN_TABLE results, you must understand how DB2 manipulates predicates. The information in Table 101 on page 719 is also helpful.

## Predicate modifications for IN-list predicates

If an IN-list predicate has only one item in its list, the predicate becomes an EQUAL predicate.

A set of simple, Boolean term, equal predicates on the same column that are connected by OR predicates can be converted into an IN-list predicate. For example: `C1=5 or C1=10 or C1=15` converts to `C1 IN (5,10,15)`.

## When DB2 simplifies join operations

Because full outer joins are less efficient than left or right joins, and left and right joins are less efficient than inner joins, you should always try to use the simplest type of join operation in your queries. However, if DB2 encounters a join operation that it can simplify, it attempts to do so. In general, DB2 can simplify a join operation when the query contains a predicate or an ON clause that eliminates the null values that are generated by the join operation.

For example, consider this query:

```
SELECT * FROM T1 X FULL JOIN T2 Y
  ON X.C1=Y.C1
  WHERE X.C2 > 12;
```

The outer join operation gives you these result table rows:

- The rows with matching values of C1 in tables T1 and T2 (the inner join result)
- The rows from T1 where C1 has no corresponding value in T2
- The rows from T2 where C1 has no corresponding value in T1

However, when you apply the predicate, you remove all rows in the result table that came from T2 where C1 has no corresponding value in T1. DB2 transforms the full join into a left join, which is more efficient:

```
SELECT * FROM T1 X LEFT JOIN T2 Y
  ON X.C1=Y.C1
  WHERE X.C2 > 12;
```

In the following example, the predicate, X.C2>12, filters out all null values that result from the right join:

```
SELECT * FROM T1 X RIGHT JOIN T2 Y
  ON X.C1=Y.C1
  WHERE X.C2>12;
```

Therefore, DB2 can transform the right join into a more efficient inner join without changing the result:

```
SELECT * FROM T1 X INNER JOIN T2 Y
  ON X.C1=Y.C1
  WHERE X.C2>12;
```

The predicate that follows a join operation must have the following characteristics before DB2 transforms an outer join into a simpler outer join or into an inner join:

- The predicate is a Boolean term predicate.
- The predicate is false if one table in the join operation supplies a null value for all of its columns.

These predicates are examples of predicates that can cause DB2 to simplify join operations:

- T1.C1 > 10
- T1.C1 IS NOT NULL
- T1.C1 > 10 OR T1.C2 > 15
- T1.C1 > T2.C1
- T1.C1 IN (1,2,4)
- T1.C1 LIKE 'ABC%'
- T1.C1 BETWEEN 10 AND 100
- 12 BETWEEN T1.C1 AND 100

The following example shows how DB2 can simplify a join operation because the query contains an ON clause that eliminates rows with unmatched values:

```
SELECT * FROM T1 X LEFT JOIN T2 Y
  FULL JOIN T3 Z ON Y.C1=Z.C1
  ON X.C1=Y.C1;
```

Because the last ON clause eliminates any rows from the result table for which column values that come from T1 or T2 are null, DB2 can replace the full join with a more efficient left join to achieve the same result:

```
SELECT * FROM T1 X LEFT JOIN T2 Y
  LEFT JOIN T3 Z ON Y.C1=Z.C1
  ON X.C1=Y.C1;
```

There is one case in which DB2 transforms a full outer join into a left join when you cannot write code to do it. This is the case where a view specifies a full outer join, but a subsequent query on that view requires only a left outer join. For example, consider this view:

```
CREATE VIEW V1 (C1,T1C2,T2C2) AS
  SELECT COALESCE(T1.C1, T2.C1), T1.C2, T2.C2
  FROM T1 X FULL JOIN T2 Y
  ON T1.C1=T2.C1;
```

This view contains rows for which values of C2 that come from T1 are null. However, if you execute the following query, you eliminate the rows with null values for C2 that come from T1:

```
SELECT * FROM V1
  WHERE T1C2 > 10;
```

Therefore, for this query, a left join between T1 and T2 would have been adequate. DB2 can execute this query as if the view V1 was generated with a left outer join so that the query runs more efficiently.

## Predicates generated through transitive closure

When the set of predicates that belong to a query logically imply other predicates, DB2 can generate additional predicates to provide more information for access path selection.

***Rules for generating predicates:*** For single-table or inner join queries, DB2 generates predicates for transitive closure if:

- The query has an equal type predicate: COL1=COL2. This could be:
  - A local predicate
  - A join predicate
- The query also has a Boolean term predicate on one of the columns in the first predicate with one of the following formats:
  - COL1 *op value*

    *op* is =, <>, >, >=, <, or <=.

    *value* is a constant, host variable, or special register.
  - COL1 (NOT) BETWEEN *value1* AND *value2*
  - COL1=COL3

For outer join queries, DB2 generates predicates for transitive closure if the query has an ON clause of the form COL1=COL2 and a before join predicate that has one of the following formats:

- COL1 *op value*

  *op* is =, <>, >, >=, <, or <=
- COL1 (NOT) BETWEEN *value1* AND *value2*

DB2 generates a transitive closure predicate for an outer join query only if the generated predicate does not reference the table with unmatched rows. That is, the generated predicate cannot reference the left table for a left outer join or the right table for a right outer join.

When a predicate meets the the transitive closure conditions, DB2 generates a new predicate, whether or not it already exists in the WHERE clause.

The generated predicates have one of the following formats:

- COL *op value*

  *op* is =, <>, >, >=, <, or <=.

  *value* is a constant, host variable, or special register.
- COL (NOT) BETWEEN *value1* AND *value2*
- COL1=COL2 (for single-table or inner join queries only)

***Example of transitive closure for an inner join:*** Suppose that you have written this query, which meets the conditions for transitive closure:

```
SELECT * FROM T1, T2
  WHERE T1.C1=T2.C1 AND
  T1.C1>10;
```

DB2 generates an additional predicate to produce this query, which is more efficient:

```
SELECT * FROM T1, T2
  WHERE T1.C1=T2.C1 AND
  T1.C1>10 AND
  T2.C1>10;
```

***Example of transitive closure for an outer join:*** Suppose that you have written this outer join query:

```
SELECT * FROM (SELECT * FROM T1 WHERE T1.C1>10) X
  LEFT JOIN T2
  ON X.C1 = T2.C1;
```

The before join predicate, T1.C1>10, meets the conditions for transitive closure, so DB2 generates this query:

```
SELECT * FROM
  (SELECT * FROM T1 WHERE T1.C1>10 AND T2.C1>10) X
  LEFT JOIN T2
  ON X.C1 = T2.C1;
```

***Predicate redundancy:*** A predicate is redundant if evaluation of other predicates in the query already determines the result that the predicate provides. You can specify redundant predicates or DB2 can generate them. DB2 does not determine that any of your query predicates are redundant. All predicates that you code are evaluated at execution time regardless of whether they are redundant. If DB2 generates a redundant predicate to help select access paths, that predicate is ignored at execution.

***Adding extra predicates:*** DB2 performs predicate transitive closure only on equal and range predicates. Other types of predicates, such as IN or LIKE predicates, might be needed in the following case:

```
SELECT * FROM T1,T2
   WHERE T1.C1=T2.C1
     AND T1.C1 LIKE 'A%';
```

In this case, add the predicate T2.C1 LIKE ’A%’.

## Column correlation

Two columns of data, A and B of a single table, are correlated if the values in column A do not vary independently of the values in column B.

The following is an excerpt from a large single table. Columns CITY and STATE are highly correlated, and columns DEPTNO and SEX are entirely independent.

***TABLE CREWINFO***

| CITY | STATE | DEPTNO | SEX | EMPNO | ZIPCODE |
|---|---|---|---|---|---|
| Fresno | CA | A345 | F | 27375 | 93650 |
| Fresno | CA | J123 | M | 12345 | 93710 |
| Fresno | CA | J123 | F | 93875 | 93650 |
| Fresno | CA | J123 | F | 52325 | 93792 |
| New York | NY | J123 | M | 19823 | 09001 |
| New York | NY | A345 | M | 15522 | 09530 |
| Miami | FL | B499 | M | 83825 | 33116 |
| Miami | FL | A345 | F | 35785 | 34099 |
| Los Angeles | CA | X987 | M | 12131 | 90077 |
| Los Angeles | CA | A345 | M | 38251 | 90091 |

In this simple example, for every value of column CITY that equals 'FRESNO', there is the same value in column STATE ('CA').

### How to detect column correlation

The first indication that column correlation is a problem is because of poor response times when DB2 has chosen an inappropriate access path. If you suspect two columns in a table (CITY and STATE in table CREWINFO) are correlated, then you can issue the following SQL queries that reflect the relationships between the columns:

```
SELECT COUNT (DISTINCT CITY) FROM CREWINFO; (RESULT1)
SELECT COUNT (DISTINCT STATE) FROM CREWINFO;  (RESULT2)
```

The result of the count of each distinct column is the value of COLCARDF in the DB2 catalog table SYSCOLUMNS. Multiply the above two values together to get a preliminary result:

```
RESULT1  x  RESULT2    =      ANSWER1
```

Then issue the following SQL statement:

```
SELECT COUNT(*) FROM
  (SELECT DISTINCT CITY,STATE
   FROM CREWINFO) AS V1;          (ANSWER2)
```

Compare the result of the above count (ANSWER2) with ANSWER1. If ANSWER2 is less than ANSWER1, then the suspected columns are correlated.

## Impacts of column correlation

DB2 might not determine the best access path, table order, or join method when your query uses columns that are highly correlated. Column correlation can make the estimated cost of operations cheaper than they actually are. Column correlation affects both single table queries and join queries.

***Column correlation on the best matching columns of an index:*** The following query selects rows with females in department A345 from Fresno, California. There are 2 indexes defined on the table, Index 1 (CITY,STATE,ZIPCODE) and Index 2 (DEPTNO,SEX).

```
Query 1

SELECT ... FROM CREWINFO WHERE
  CITY = 'FRESNO' AND STATE = 'CA'         (PREDICATE1)
  AND DEPTNO = 'A345' AND SEX = 'F';       (PREDICATE2)
```

Consider the two compound predicates (labeled PREDICATE1 and PREDICATE2), their actual filtering effects (the proportion of rows they select), and their DB2 filter factors. Unless the proper catalog statistics are gathered, the filter factors are calculated as if the columns of the predicate are entirely independent (not correlated).

*Table 106. Effects of column correlation on matching columns*

| | INDEX 1 | INDEX 2 |
|---|---|---|
| Matching Predicates | Predicate1<br>CITY=FRESNO AND STATE=CA | Predicate2<br>DEPTNO=A345 AND SEX=F |
| Matching Columns | 2 | 2 |
| DB2 estimate for matching columns (Filter Factor) | column=CITY, COLCARDF=4<br>Filter Factor=1/4<br>column=STATE, COLCARDF=3<br>Filter Factor=1/3 | column=DEPTNO, COLCARDF=4<br>Filter Factor=1/4<br>column=SEX, COLCARDF=2<br>Filter Factor=1/2 |
| Compound Filter Factor for matching columns | 1/4 × 1/3 = 0.083 | 1/4 × 1/2 = 0.125 |
| Qualified leaf pages based on DB2 estimations | 0.083 × 10 = 0.83<br>INDEX CHOSEN (.8 < 1.25) | 0.125 × 10 = 1.25 |
| Actual filter factor based on data distribution | 4/10 | 2/10 |
| Actual number of qualified leaf pages based on compound predicate | 4/10 × 10 = 4 | 2/10 × 10 = 2<br>BETTER INDEX CHOICE<br>(2 < 4) |

DB2 chooses an index that returns the fewest rows, partly determined by the smallest filter factor of the matching columns. Assume that filter factor is the only influence on the access path. The combined filtering of columns CITY and STATE seems very good, whereas the matching columns for the second index do not seem to filter as much. Based on those calculations, DB2 chooses Index 1 as an access path for Query 1.

The problem is that the filtering of columns CITY and STATE should not look good. Column STATE does almost no filtering. Since columns DEPTNO and SEX do a better job of filtering out rows, DB2 should favor Index 2 over Index 1.

***Column correlation on index screening columns of an index:*** Correlation might also occur on nonmatching index columns, used for index screening. See "Nonmatching index scan (ACCESSTYPE=I and MATCHCOLS=0)" on page 809 for more information. Index screening predicates help reduce the number of data rows that qualify while scanning the index. However, if the index screening predicates are correlated, they do not filter as many data rows as their filter factors suggest. To illustrate this, use the same Query 1 (see page 732) with the following indexes on table CREWINFO (page 731):

```
Index 3 (EMPNO,CITY,STATE)
Index 4 (EMPNO,DEPTNO,SEX)
```

In the case of Index 3, because the columns CITY and STATE of Predicate 1 are correlated, the index access is not improved as much as estimated by the screening predicates and therefore Index 4 might be a better choice. (Note that index screening also occurs for indexes with matching columns greater than zero.)

***Multiple table joins:*** In Query 2, an additional table is added to the original query (see Query 1 on page 732) to show the impact of column correlation on join queries.

```
TABLE DEPTINFO

CITY          STATE    MANAGER     DEPT    DEPTNAME
----------------------------------------------------
FRESNO        CA       SMITH       J123     ADMIN
LOS ANGELES   CA       JONES       A345     LEGAL
```

```
Query 2
SELECT ... FROM CREWINFO T1,DEPTINFO T2
   WHERE T1.CITY = 'FRESNO' AND T1.STATE='CA'          (PREDICATE 1)
   AND T1.DEPTNO = T2.DEPT AND T2.DEPTNAME = 'LEGAL';
```

The order that tables are accessed in a join statement affects performance. The estimated combined filtering of Predicate1 is lower than its actual filtering. So table CREWINFO might look better as the first table accessed than it should.

Also, due to the smaller estimated size for table CREWINFO, a nested loop join might be chosen for the join method. But, if many rows are selected from table CREWINFO because Predicate1 does not filter as many rows as estimated, then another join method might be better.

## What to do about column correlation

If column correlation is causing DB2 to choose an inappropriate access path, try one of these techniques to alter the access path:

- If the correlated columns are concatenated key columns of an index, run the utility RUNSTATS with options KEYCARD and FREQVAL. This is the preferred technique.

- Update the catalog statistics manually.
- Use SQL that forces access through a particular index.

The last two techniques are discussed in “Special techniques to influence access path selection” on page 746.

The utility RUNSTATS collects the statistics DB2 needs to make proper choices about queries. With RUNSTATS, you can collect statistics on the concatenated key columns of an index and the number of distinct values for those concatenated columns. This gives DB2 accurate information to calculate the filter factor for the query.

For example, RUNSTATS collects statistics that benefit queries like this:

```
SELECT * FROM T1
 WHERE C1 = 'a' AND C2 = 'b' AND C3 = 'c' ;
```

where:
- The first three index keys are used (MATCHCOLS = 3).
- An index exists on C1, C2, C3, C4, C5.
- Some or all of the columns in the index are correlated in some way.

See “Use RUNSTATS to keep access path statistics current” on page 537 for information on using RUNSTATS to influence access path selection. See “Updating catalog statistics” on page 754 for information on updating catalog statistics manually.

## Using host variables efficiently

***Host variables require default filter factors:*** When you bind a static SQL statement that contains host variables, DB2 uses a default filter factor to determine the best access path for the SQL statement. For more information on filter factors, including default values, see “Predicate filter factors” on page 723.

DB2 often chooses an access path that performs well for a query with several host variables. However, in a new release or after maintenance has been applied, DB2 might choose a new access path that does not perform as well as the old access path. In most cases, the change in access paths is due to the default filter factors, which might lead DB2 to optimize the query in a different way.

There are two ways to change the access path for a query that contains host variables:

- Bind the package or plan that contains the query with the option REOPT(VARS).
- Rewrite the query.

## Using REOPT(VARS) to change the access path at run time

Specify the bind option REOPT(VARS) when you want DB2 to determine access paths at both bind time and run time for statements that contain one or more of the following:
- host variables
- parameter markers
- special registers

At run time, DB2 uses the values in those variables to determine the access paths.

Because there is a performance cost to reoptimizing the access path at run time, you should use the bind option REOPT(VARS) only on packages or plans containing statements that perform poorly.

Be careful when using REOPT(VARS) for a statement executed in a loop; the reoptimization occurs with every execution of that statement. However, if you are using a cursor, you can put the FETCH statements in a loop because the reoptimization only occurs when the cursor is opened.

To use REOPT(VARS) most efficiently, first determine which SQL statements in your applications perform poorly. Separate the code containing those statements into units that you bind into packages with the option REOPT(VARS). Bind the rest of the code into packages using NOREOPT(VARS). Then bind the plan with the option NOREOPT(VARS). Only statements in the packages bound with REOPT(VARS) are candidates for reoptimization at run time.

To determine which queries in plans and packages bound with REOPT(VARS) will be reoptimized at run time, execute the following SELECT statements:

```
SELECT PLNAME,
  CASE WHEN STMTNOI <> 0
   THEN STMTNOI
   ELSE STMTNO
  END AS STMTNUM,
  SEQNO, TEXT
  FROM   SYSIBM.SYSSTMT
  WHERE STATUS IN ('B','F','G','J')
  ORDER BY PLNAME, STMTNUM, SEQNO;
```

```
SELECT COLLID, NAME, VERSION,
  CASE WHEN STMTNOI <> 0
   THEN STMTNOI
   ELSE STMTNO
  END AS STMTNUM,
  SEQNO, STMT
  FROM SYSIBM.SYSPACKSTMT
  WHERE STATUS IN ('B','F','G','J')
  ORDER BY COLLID, NAME, VERSION, STMTNUM, SEQNO;
```

If you specify the bind option VALIDATE(RUN), and a statement in the plan or package is not bound successfully, that statement is incrementally bound at run time. If you also specify the bind option REOPT(VARS), DB2 reoptimizes the access path during the incremental bind.

To determine which plans and packages have statements that will be incrementally bound, execute the following SELECT statements:

```
SELECT DISTINCT NAME
  FROM SYSIBM.SYSSTMT
  WHERE STATUS = 'F' OR STATUS = 'H';
```

```
SELECT DISTINCT COLLID, NAME, VERSION
  FROM   SYSIBM.SYSPACKSTMT
  WHERE STATUS = 'F' OR STATUS = 'H';
```

## Rewriting queries to influence access path selection

The examples that follow identify potential performance problems and offer suggestions for tuning the queries. However, before you rewrite any query, you should consider whether the bind option REOPT(VARS) can solve your access path problems. See “Using REOPT(VARS) to change the access path at run time” on page 734 for more information on REOPT(VARS).

***Example 1: An equal predicate***

An equal predicate has a default filter factor of 1/COLCARDF. The actual filter factor might be quite different.

***Query:***

```
     SELECT * FROM DSN8710.EMP
      WHERE SEX = :HV1;
```

***Assumptions:*** Because there are only two different values in column SEX, ’M’ and ’F’, the value COLCARDF for SEX is 2. If the numbers of male and female employees are not equal, the actual filter factor of 1/2 is larger or smaller than the default, depending on whether `:HV1` is set to ’M’ or ’F’.

***Recommendation:*** One of these two actions can improve the access path:

- Bind the package or plan that contains the query with the option REOPT(VARS). This action causes DB2 to reoptimize the query at run time, using the input values you provide.
- Write predicates to influence DB2's selection of an access path, based on your knowledge of actual filter factors. For example, you can break the query above into three different queries, two of which use constants. DB2 can then determine the exact filter factor for most cases when it binds the plan.

```
SELECT (HV1);
  WHEN ('M')
    DO;
      EXEC SQL SELECT * FROM DSN8710.EMP
       WHERE SEX = 'M';
    END;
  WHEN ('F')
    DO;
      EXEC SQL SELECT * FROM DSN8710.EMP
       WHERE SEX = 'F';
    END;
  OTHERWISE
    DO:
      EXEC SQL SELECT * FROM DSN8710.EMP
       WHERE SEX = :HV1;
    END;
END;
```

***Example 2: Known ranges***

Table T1 has two indexes: T1X1 on column C1 and T1X2 on column C2.

***Query:***

```
SELECT * FROM T1
       WHERE C1 BETWEEN :HV1 AND :HV2
         AND C2 BETWEEN :HV3 AND :HV4;
```

***Assumptions:*** You know that:

- The application always provides a narrow range on C1 and a wide range on C2.
- The desired access path is through index T1X1.

***Recommendation:*** If DB2 does not choose T1X1, rewrite the query as follows, so that DB2 does not choose index T1X2 on C2:

```
SELECT * FROM T1
       WHERE C1 BETWEEN :HV1 AND :HV2
         AND (C2 BETWEEN :HV3 AND :HV4 OR 0=1);
```

### *Example 3: Variable ranges*

Table T1 has two indexes: T1X1 on column C1 and T1X2 on column C2.

***Query:***

```
SELECT * FROM T1
       WHERE C1 BETWEEN :HV1 AND :HV2
         AND C2 BETWEEN :HV3 AND :HV4;
```

***Assumptions:*** You know that the application provides both narrow and wide ranges on C1 and C2. Hence, default filter factors do not allow DB2 to choose the best access path in all cases. For example, a small range on C1 favors index T1X1 on C1, a small range on C2 favors index T1X2 on C2, and wide ranges on both C1 and C2 favor a table space scan.

***Recommendation:*** If DB2 does not choose the best access path, try either of the following changes to your application:

- Use a dynamic SQL statement and embed the ranges of C1 and C2 in the statement. With access to the actual range values, DB2 can estimate the actual filter factors for the query. Preparing the statement each time it is executed requires an extra step, but it can be worthwhile if the query accesses a large amount of data.
- Include some simple logic to check the ranges of C1 and C2, and then execute one of these static SQL statements, based on the ranges of C1 and C2:

```
SELECT * FROM T1 WHERE C1 BETWEEN :HV1 AND :HV2
                AND  (C2 BETWEEN :HV3 AND :HV4 OR 0=1);

SELECT * FROM T1 WHERE C2 BETWEEN :HV3 AND :HV4
                AND  (C1 BETWEEN :HV1 AND :HV2 OR 0=1);

SELECT * FROM T1 WHERE (C1 BETWEEN :HV1 AND :HV2 OR 0=1)
                AND   (C2 BETWEEN :HV3 AND :HV4 OR 0=1);
```

### *Example 4: ORDER BY*

Table T1 has two indexes: T1X1 on column C1 and T1X2 on column C2.

***Query:***

```
SELECT * FROM T1
       WHERE C1 BETWEEN :HV1 AND :HV2
       ORDER BY C2;
```

In this example, DB2 could choose one of the following actions:

- Scan index T1X1 and then sort the results by column C2
- Scan the table space in which T1 resides and then sort the results by column C2
- Scan index T1X2 and then apply the predicate to each row of data, thereby avoiding the sort

Which choice is best depends on the following factors:

- The number of rows that satisfy the range predicate
- Which index has the higher cluster ratio

If the actual number of rows that satisfy the range predicate is significantly different from the estimate, DB2 might not choose the best access path.

***Assumptions:*** You disagree with DB2’s choice.

***Recommendation:*** In your application, use a dynamic SQL statement and embed the range of C1 in the statement. That allows DB2 to use the actual filter factor rather than the default, but requires extra processing for the PREPARE statement.

***Example 5: A join operation***

Tables A, B, and C each have indexes on columns C1, C2, C3, and C4.

***Query:***

```
SELECT * FROM A, B, C
       WHERE A.C1 = B.C1
         AND A.C2 = C.C2
         AND A.C2 BETWEEN :HV1 AND :HV2
         AND A.C3 BETWEEN :HV3 AND :HV4
         AND A.C4 < :HV5
         AND B.C2 BETWEEN :HV6 AND :HV7
         AND B.C3 < :HV8
         AND C.C2 < :HV9;
```

***Assumptions:*** The actual filter factors on table A are much larger than the default factors. Hence, DB2 underestimates the number of rows selected from table A and wrongly chooses that as the first table in the join.

***Recommendations:*** You can:

- Reduce the estimated size of Table A by adding predicates
- Disfavor any index on the join column by making the join predicate on table A nonindexable

The query below illustrates the second of those choices.

```
SELECT * FROM T1 A, T1 B, T1 C
       WHERE (A.C1 = B.C1 OR 0=1)
         AND A.C2 = C.C2
         AND A.C2 BETWEEN :HV1 AND :HV2
         AND A.C3 BETWEEN :HV3 AND :HV4
         AND A.C4 < :HV5
         AND B.C2 BETWEEN :HV6 AND :HV7
         AND B.C3 < :HV8
         AND C.C2 < :HV9;
```

The result of making the join predicate between A and B a nonindexable predicate (which cannot be used in single index access) disfavors the use of the index on column C1. This, in turn, might lead DB2 to access table A or B first. Or, it might lead DB2 to change the access type of table A or B, thereby influencing the join sequence of the other tables.

## Writing efficient subqueries

***Definitions:*** A *subquery* is a SELECT statement within the WHERE or HAVING clause of another SQL statement.

***Decision needed:*** You can often write two or more SQL statements that achieve identical results, particularly if you use subqueries. The statements have different access paths, however, and probably perform differently.

***Topic overview:*** The topics that follow describe different methods to achieve the results intended by a subquery and tell what DB2 does for each method. The information should help you estimate what method performs best for your query.

The first two methods use different types of subqueries:

- “Correlated subqueries”
- “Noncorrelated subqueries” on page 740

A subquery can sometimes be transformed into a join operation. Sometimes DB2 does that to improve the access path, and sometimes you can get better results by doing it yourself. The third method is:

- “Subquery transformation into join” on page 741

  Finally, for a comparison of the three methods as applied to a single task, see:
- “Subquery tuning” on page 743

## Correlated subqueries

***Definition:*** A *correlated* subquery refers to at least one column of the outer query.

Any predicate that contains a correlated subquery is a stage 2 predicate.

***Example:*** In the following query, the correlation name, X, illustrates the subquery’s reference to the outer query block.

```
SELECT * FROM DSN8710.EMP X
   WHERE  JOB = 'DESIGNER'
      AND  EXISTS (SELECT 1
                  FROM   DSN8710.PROJ
                  WHERE  DEPTNO = X.WORKDEPT
                     AND  MAJPROJ = 'MA2100');
```

***What DB2 does:*** A correlated subquery is evaluated for each qualified row of the outer query that is referred to. In executing the example, DB2:

1. Reads a row from table EMP where JOB=’DESIGNER’.
2. Searches for the value of WORKDEPT from that row, in a table stored in memory.

   The in-memory table saves executions of the subquery. If the subquery has already been executed with the value of WORKDEPT, the result of the subquery is in the table and DB2 does not execute it again for the current row. Instead, DB2 can skip to step 5.
3. Executes the subquery, if the value of WORKDEPT is not in memory. That requires searching the PROJ table to check whether there is any project, where MAJPROJ is ’MA2100’, for which the current WORKDEPT is responsible.
4. Stores the value of WORKDEPT and the result of the subquery in memory.
5. Returns the values of the current row of EMP to the application.

DB2 repeats this whole process for each qualified row of the EMP table.

***Notes on the in-memory table:*** The in-memory table is applicable if the operator of the predicate that contains the subquery is one of the following operators:

<, <=, >, >=, =, <>, EXISTS, NOT EXISTS

The table is not used, however, if:

- There are more than 16 correlated columns in the subquery
- The sum of the lengths of the correlated columns is more than 256 bytes
- There is a unique index on a subset of the correlated columns of a table from the outer query

The in-memory table is a wrap-around table and does not guarantee saving the results of all possible duplicated executions of the subquery.

## Noncorrelated subqueries

***Definition:*** A *noncorrelated* subquery makes no reference to outer queries.

***Example:***

```
SELECT * FROM DSN8710.EMP
  WHERE  JOB = 'DESIGNER'
    AND  WORKDEPT IN (SELECT DEPTNO
                      FROM   DSN8710.PROJ
                      WHERE  MAJPROJ = 'MA2100');
```

***What DB2 does:*** A noncorrelated subquery is executed once when the cursor is opened for the query. What DB2 does to process it depends on whether it returns a single value or more than one value. The query in the example above can return more than one value.

### Single-value subqueries

When the subquery is contained in a predicate with a simple operator, the subquery is required to return 1 or 0 rows. The simple operator can be one of the following operators:

<, <=, >, >=, =, <>, EXISTS, NOT EXISTS

The following noncorrelated subquery returns a single value:

```
  SELECT *
  FROM   DSN8710.EMP
  WHERE  JOB = 'DESIGNER'
    AND  WORKDEPT <= (SELECT MAX(DEPTNO)
                     FROM   DSN8710.PROJ);
```

***What DB2 does:*** When the cursor is opened, the subquery executes. If it returns more than one row, DB2 issues an error. The predicate that contains the subquery is treated like a simple predicate with a constant specified, for example, WORKDEPT <= *'value'*.

***Stage 1 and stage 2 processing:*** The rules for determining whether a predicate with a noncorrelated subquery that returns a single value is stage 1 or stage 2 are generally the same as for the same predicate with a single variable. However, the predicate is stage 2 if:

- The value returned by the subquery is nullable and the column of the outer query is not nullable.
- The data type of the subquery is higher than that of the column of the outer query. For example, the following predicate is stage 2:

  ```
  WHERE SMALLINT_COL < (SELECT INTEGER_COL FROM ...
  ```

### Multiple-value subqueries

A subquery can return more than one value if the operator is one of the following:

*op* ANY *op* ALL *op* SOME IN EXISTS

where *op* is any of the operators >, >=, <, or <=.

***What DB2 does:*** If possible, DB2 reduces a subquery that returns more than one row to one that returns only a single row. That occurs when there is a range comparison along with ANY, ALL, or SOME. The following query is an example:

```
SELECT * FROM DSN8710.EMP
   WHERE  JOB = 'DESIGNER'
     AND  WORKDEPT <= ANY (SELECT DEPTNO
                          FROM   DSN8710.PROJ
                          WHERE  MAJPROJ = 'MA2100');
```

DB2 calculates the maximum value for DEPTNO from table DSN8710.PROJ and removes the ANY keyword from the query. After this transformation, the subquery is treated like a single-value subquery.

That transformation can be made with a *maximum value* if the range operator is:

- > or >= with the quantifier ALL
- < or <= with the quantifier ANY or SOME

The transformation can be made with a *minimum value* if the range operator is:

- < or <= with the quantifier ALL
- > or >= with the quantifier ANY or SOME

The resulting predicate is determined to be stage 1 or stage 2 by the same rules as for the same predicate with a single-valued subquery.

***When a subquery is sorted:*** A noncorrelated subquery is sorted in descending order when the comparison operator is IN, NOT IN, = ANY, <> ANY, = ALL, or <> ALL. The sort enhances the predicate evaluation, reducing the amount of scanning on the subquery result. When the value of the subquery becomes smaller or equal to the expression on the left side, the scanning can be stopped and the predicate can be determined to be true or false.

When the subquery result is a character data type and the left side of the predicate is a datetime data type, then the result is placed in a work file without sorting. For some noncorrelated subqueries using the above comparison operators, DB2 can more accurately pinpoint an entry point into the work file, thus further reducing the amount of scanning that is done.

***Results from EXPLAIN:*** For information about the result in a plan table for a subquery that is sorted, see “When are column functions evaluated? (COLUMN_FN_EVAL)” on page 805.

## Subquery transformation into join

| For a SELECT, UPDATE, or DELETE statement, DB2 can sometimes transform a subquery into a join between the result table of a subquery and the result table of an outer query.

For a SELECT statement, DB2 does the transformation if the following conditions are true:

- The transformation does not introduce redundancy.
- The subquery appears in a WHERE clause.
- The subquery does not contain GROUP BY, HAVING, or column functions.
- The subquery has only one table in the FROM clause.
- The transformation results in 15 or fewer tables in the join.
- The subquery select list has only one column, guaranteed by a unique index to have unique values.
- The comparison operator of the predicate containing the subquery is IN, = ANY, or = SOME.

- For a noncorrelated subquery, the left side of the predicate is a single column with the same data type and length as the subquery's column. (For a correlated subquery, the left side can be any expression.)

For an UPDATE or DELETE statement, or a SELECT statement that does not meet the previous conditions for transformation, DB2 does the transformation of a correlated subquery into a join if the following conditions are true:

- The transformation does not introduce redundancy.
- The subquery is correlated to its immediate outer query.
- The FROM clause of the subquery contains only one table, and the outer query (for SELECT), UPDATE, or DELETE references only one table.
- If the outer predicate is a quantified predicate with an operator of =ANY or an IN predicate, the following conditions are true:
  - The left side of the outer predicate is a single column.
  - The right side of the outer predicate is a subquery that references a single column.
  - The two columns have the same data type and length.
- The subquery does not contain the GROUP BY or DISTINCT clauses.
- The subquery does not contain column functions.
- The SELECT clause of the subquery does not contain a user-defined function with an external action or a user-defined function that modifies data.
- The subquery predicate is a Boolean term predicate.
- The predicates in the subquery that provide correlation are stage 1 predicates.
- The subquery does not contain nested subqueries.
- The subquery does not contain a self-referencing UPDATE or DELETE.
- For a SELECT statement, the query does not contain the FOR UPDATE OF clause.
- For an UPDATE or DELETE statement, the statement is a searched UPDATE or DELETE.
- For a SELECT statement, parallelism is not enabled.

For a statement with multiple subqueries, DB2 does the transformation only on the last subquery in the statement that qualifies for transformation.

***Example:*** The following subquery can be transformed into a join because it meets the first set of conditions for transformation:

```
SELECT * FROM EMP
  WHERE DEPTNO IN
   (SELECT DEPTNO FROM DEPT
      WHERE LOCATION IN ('SAN JOSE', 'SAN FRANCISCO')
      AND DIVISION = 'MARKETING');
```

If there is a department in the marketing division which has branches in both San Jose and San Francisco, the result of the above SQL statement is not the same as if a join were done. The join makes each employee in this department appear twice because it matches once for the department of location San Jose and again of location San Francisco, although it is the same department. Therefore, it is clear that to transform a subquery into a join, the uniqueness of the subquery select list must be guaranteed. For this example, a unique index on any of the following sets of columns would guarantee uniqueness:

- (DEPTNO)
- (DIVISION, DEPTNO)

- (DEPTNO, DIVISION).

The resultant query is:

```
SELECT EMP.* FROM EMP, DEPT
   WHERE EMP.DEPTNO = DEPT.DEPTNO AND
         DEPT.LOCATION IN ('SAN JOSE', 'SAN FRANCISCO') AND
         DEPT.DIVISION = 'MARKETING';
```

***Example:*** The following subquery can be transformed into a join because it meets the second set of conditions for transformation:

```
UPDATE T1 SET T1.C1 = 1
  WHERE T1.C1 =ANY
    (SELECT T2.C1 FROM T2
      WHERE T2.C2 = T1.C2);
```

***Results from EXPLAIN:*** For information about the result in a plan table for a subquery that is transformed into a join operation, see “Is a subquery transformed into a join?” on page 805.

## Subquery tuning

The following three queries all retrieve the same rows. All three retrieve data about all designers in departments that are responsible for projects that are part of major project MA2100. These three queries show that there are several ways to retrieve a desired result.

### Query A: A join of two tables

```
SELECT DSN8710.EMP.* FROM DSN8710.EMP, DSN8710.PROJ
   WHERE  JOB = 'DESIGNER'
     AND  WORKDEPT = DEPTNO
     AND  MAJPROJ = 'MA2100';
```

### Query B: A correlated subquery

```
SELECT * FROM DSN8710.EMP X
   WHERE  JOB = 'DESIGNER'
     AND  EXISTS (SELECT 1 FROM DSN8710.PROJ
                  WHERE  DEPTNO = X.WORKDEPT
                    AND  MAJPROJ = 'MA2100');
```

### Query C: A noncorrelated subquery

```
SELECT * FROM DSN8710.EMP
    WHERE  JOB = 'DESIGNER'
      AND  WORKDEPT IN (SELECT DEPTNO FROM DSN8710.PROJ
                        WHERE  MAJPROJ = 'MA2100');
```

If you need columns from both tables EMP and PROJ in the output, you must use a join.

PROJ might contain duplicate values of DEPTNO in the subquery, so that an equivalent join cannot be written.

In general, query A might be the one that performs best. However, if there is no index on DEPTNO in table PROJ, then query C might perform best. The IN-subquery predicate in query C is indexable. Therefore, if an index on WORKDEPT exists, DB2 might do IN-list access on table EMP. If you decide that a join cannot be used and there is an available index on DEPTNO in table PROJ, then query B might perform best.

When looking at a problem subquery, see if the query can be rewritten into another format or see if there is an index that you can create to help improve the performance of the subquery.

It is also important to know the sequence of evaluation, for the different subquery predicates as well as for all other predicates in the query. If the subquery predicate is costly, perhaps another predicate could be evaluated before that predicate so that the rows would be rejected before even evaluating the problem subquery predicate.

## Using scrollable cursors efficiently

The following recommendations help you get the best performance from your scrollable cursors:

- Determine when scrollable cursors work best for you.

  Scrollable cursors are a valuable tool for writing applications such as screen-based applications, in which the result table is small and you often move back and forth through the data. However, scrollable cursors require more DB2 processing than non-scrollable cursors. If your applications require large result tables or you only need to move sequentially forward through the data, use non-scrollable cursors.
- Declare scrollable cursors as SENSITIVE only if you need to see the latest data.

  If you do not need to see updates that are made by other cursors or application processes, using a cursor that you declare as INSENSITIVE requires less processing by DB2.
- To ensure maximum concurrency when you use a scrollable cursor for positioned update and delete operations, specify ISOLATION(CS) and CURRENTDATA(NO) when you bind packages and plans that contain updatable scrollable cursors. See “Chapter 30. Improving concurrency” on page 643 for more details.
- Use the FETCH FIRST *n* ROWS ONLY clause with scrollable cursors when it is appropriate.

  In a distributed environment, when you need to retrieve a limited number of rows, FETCH FIRST *n* ROWS ONLY can improve your performance for distributed queries that use DRDA access by eliminating unneeded network traffic. See Part 4 of *DB2 Application Programming and SQL Guide* for more information.

  In a local environment, if you need to scroll through a limited subset of rows in a table, you can use FETCH FIRST *n* ROWS ONLY to make the result table smaller.
- In a distributed environment, if you do not need to use your scrollable cursors to modify data, do your cursor processing in a stored procedure.

  Using stored procedures can decrease the amount of network traffic that your application requires.
- Create TEMP table spaces that are large enough to process your scrollable cursors.

  See Part 2 of *DB2 Installation Guide* for information on calculating the appropriate size for declared temporary tables that you use for scrollable cursors.
- Remember to commit changes often.

  Because you often leave scrollable cursors open longer than non-scrollable cursors, it is important to commit changes often enough. Declare your scrollable cursors WITH HOLD to prevent the cursors from closing after a commit operation.

## Writing efficient queries on views with UNION operators

Creating views using a UNION ALL statement to combine a number of tables can be useful in a number of situations. For example:

- When a table becomes very large, it can be useful to break the table into a set of smaller tables. You can then create indexes on each of the smaller tables so that you can query each of those tables efficiently. You can also create a view that uses UNION or UNION ALL operators to logically combine the smaller tables, and then query the view as if it were the original large table.

  For example, the following definition creates a view that concatenates information from three smaller tables:

```
CREATE VIEW FIRSTQTR (SNO,CHARGES,DATE) AS
  SELECT SNO,CHARGES,DATE
   FROM MONTH1
   WHERE DATE BETWEEN '1/1/2001' AND '1/31/2001'
 UNION ALL
  SELECT SNO,CHARGES,DATE
   FROM MONTH2
   WHERE DATE BETWEEN '2/1/2001' AND '2/28/2001'
 UNION ALL
  SELECT SNO,CHARGES,DATE
   FROM MONTH3
   WHERE DATE BETWEEN '3/1/2001' AND '3/31/2001';
```

*Figure 92. Example of a view with UNION ALL operators and efficient predicates*

- A view provides a global picture of similar information from unlike tables.

  For example, suppose that you want income information about all the employees in a company. This information is stored in separate tables for executives, salespeople, and contractors. In addition, the sources of income are different for each category of employee. Executives receive a salary plus a bonus, salespeople receive a salary plus a commission, and contractors receive an hourly wage. You might use a view like this to get combined income information:

```
CREATE VIEW EMPLOYEEPAY (EMPNO, FIRSTNAME, LASTNAME, DEPTNO, YEARMONTH, TOTALPAY) AS
  (SELECT EMPNO, FIRSTNAME, LASTNAME, DEPTNO, YEARMONTH, SALARY/12 + BONUS
     FROM   EXECUTIVES
   UNION ALL
   SELECT EMPNO, FIRSTNAME, LASTNAME, DEPTNO, YEARMONTH, BASEMONSALARY+TOTALCOM
     FROM SALESPERSON S, (SELECT EMPNO, YEARMONTH, SUM(COMMISSION) TOTALCOM
                            FROM   COMMISSION C
                            GROUP BY EMPNO, YEARMONTH) COM
     WHERE S.EMPNO = COM.EMPNO
   UNION ALL
   SELECT EMPNO, FIRSTNAME, LASTNAME, DEPTNO, YEARMONTH, TOTALPAY
     FROM CONTRACTOR C, (SELECT EMPNO, YEARMONTH, SUM(PAY) TOTALPAY
                          FROM   CONTRACTPAY
                          GROUP BY EMPNO, YEARMONTH) PAY
     WHERE C.EMPNO = PAY.EMPNO);
```

The following techniques can help queries on these types of views perform better. In these suggestions, S1 through S*n* represent small tables that are combined using UNION or UNION ALL operators to form view V.

- Create a clustering index on each of S1 through S*n*.

  In a typical data warehouse model, partitions in a table are in time sequence, but the data is stored in another key sequence, such as the customer number within each partition. You can simulate partitions on view V by creating cluster indexes on S1 through S*n*.

Using separate tables to simulate a single, larger partitioned table can be more flexible than using a single table. You can create different numbers and types of indexes with different clustering properties on different tables to improve performance where it is most necessary. For example, if each table represents a date range, older tables might be updated less frequently than newer tables. Therefore, for newer tables, you can create more indexes to improve query performance. In addition, if older data has different query patterns from newer data, you might want to create different clustering indexes on the tables with older and newer data so that you can reorganize the older and newer data into different orders.

- Use UNION ALL instead of UNION when they are equivalent.

  DB2 can evaluate queries that contain UNION ALL more efficiently than queries that contain UNION. Therefore, if a view produces the same result set with UNION ALL operators and UNION operators, use UNION ALL.

- Use predicates in the view definition and in queries that reference the view that let DB2 use the optimization technique of eliminating unnecessary subselects during evaluation of a query. These predicates tell DB2 about the data range of the result table for any subselect in the view. Subselects that contain the following predicates can be eliminated from query evaluation:
  - COL *op literal*

    *op* can be =, >, <, >=, <=, ¬> or ¬<
  - COL BETWEEN *literal1* AND *literal2*
  - COL IN (*literal1*, *literal2*, ...)

  DB2 can eliminate a subselect from a view only if it contains one of these predicates. Therefore, for better performance of queries that use the view, you should provide a predicate for each subselect in the view, even if a subselect is not needed to evaluate the query. For example, in Figure 92 on page 745, each table contains data for only a single month, so the BETWEEN predicate is redundant. However, when you use the UNION ALL operator and a BETWEEN predicate for every SELECT clause, DB2 can optimize queries that use the view more efficiently.

- Avoid view materialization.

  See Table 115 on page 831 for conditions under which DB2 materializes views.

## Special techniques to influence access path selection

**ATTENTION**

This section describes tactics for rewriting queries and modifying catalog statistics to influence DB2's method of selecting access paths. In a later release of DB2, the selection method might change, causing your changes to degrade performance. Save the old catalog statistics or SQL before you consider making any changes to control the choice of access path. Before and after you make any changes, take performance measurements. When you migrate to a new release, examine the performance again. Be prepared to back out any changes that have degraded performance.

This section contains the following information about determining and changing access paths:

- Obtaining information about access paths

- “Minimizing overhead for retrieving few rows: OPTIMIZE FOR n ROWS”
- “Fetching a limited number of rows: FETCH FIRST n ROWS ONLY” on page 749
- “Reducing the number of matching columns” on page 750
- “Adding extra local predicates” on page 751
- “Rearranging the order of tables in a FROM clause” on page 754
- “Updating catalog statistics” on page 754
- “Using a subsystem parameter” on page 756
- “Giving optimization hints to DB2” on page 757

## Obtaining information about access paths

There are several ways to obtain information about DB2 access paths:

- Use Visual Explain

  The DB2 Visual Explain tool, which is invoked from a workstation client, can be used to display and analyze information on access paths chosen by DB2. The tool provides you with an easy-to-use interface to the PLAN_TABLE output and allows you to invoke EXPLAIN for dynamic SQL statements. You can also access the catalog statistics for certain referenced objects of an access path. In addition, the tool allows you to archive EXPLAIN output from previous SQL statements to analyze changes in your SQL environment. See *DB2 Visual Explain online help* for more information.
- Run DB2 Performance Monitor accounting reports

  Another way to track performance is with the DB2 Performance Monitor accounting reports. The accounting report, short layout, ordered by PLANNAME, lists the primary performance figures. Check the plans that contain SQL statements whose access paths you tried to influence. If the elapsed time, TCB time, or number of getpage requests increases sharply without a corresponding increase in the SQL activity, then there could be a problem. You can use DB2 PM Online Monitor to track events after your changes have been implemented, providing immediate feedback on the effects of your changes.
- Specify the bind option EXPLAIN

  You can also use the EXPLAIN option when you bind or rebind a plan or package. Compare the new plan or package for the statement to the old one. If the new one has a table space scan or a nonmatching index space scan, but the old one did not, the problem is probably the statement. Investigate any changes in access path in the new plan or package; they could represent performance improvements or degradations. If neither the accounting report ordered by PLANNAME or PACKAGE nor the EXPLAIN statement suggest corrective action, use the DB2 PM SQL activity reports for additional information. For more information on using EXPLAIN, see “Obtaining PLAN_TABLE information from EXPLAIN” on page 790.

## Minimizing overhead for retrieving few rows: OPTIMIZE FOR n ROWS

When an application executes a SELECT statement, DB2 assumes that the application will retrieve all the qualifying rows. This assumption is most appropriate for batch environments. However, for interactive SQL applications, such as SPUFI, it is common for a query to define a very large potential result set but retrieve only the first few rows. The access path that DB2 chooses might not be optimal for those interactive applications.

This section discusses the use of OPTIMIZE FOR n ROWS to affect the performance of interactive SQL applications. Unless otherwise noted, this information pertains to local applications. For more information on using OPTIMIZE FOR n ROWS in distributed applications, see Part 4 of *DB2 Application Programming and SQL Guide*.

***What OPTIMIZE FOR n ROWS does:*** The OPTIMIZE FOR *n* ROWS clause lets an application declare its intent to do either of these things:

- Retrieve only a subset of the result set
- Give priority to the retrieval of the first few rows

DB2 uses the OPTIMIZE FOR *n* ROWS clause to choose access paths that minimize the response time for retrieving the first few rows. For distributed queries, the value of *n* determines the number of rows that DB2 sends to the client on each DRDA network transmission. See Part 4 of *DB2 Application Programming and SQL Guide* for more information on using OPTIMIZE FOR *n* ROWS in the distributed environment.

***Use OPTIMIZE FOR 1 ROW to avoid sorts:*** You can influence the access path most by using OPTIMIZE FOR 1 ROW. OPTIMIZE FOR 1 ROW tells DB2 to select an access path that returns the first qualifying row quickly. This means that whenever possible, DB2 avoids any access path that involves a sort. If you specify a value for n that is anything but 1, DB2 chooses an access path based on cost, and you won't necessarily avoid sorts.

***How to specify OPTIMIZE FOR n ROWS for a CLI application:*** For a Call Level Interface (CLI) application, you can specify that DB2 uses OPTIMIZE FOR *n* ROWS for all queries. To do that, specify the keyword OPTIMIZEFORNROWS in the initialization file. For more information, see Chapter 3 of *DB2 ODBC Guide and Reference*.

***How many rows you can retrieve with OPTIMIZE FOR n ROWS:*** The OPTIMIZE FOR *n* ROWS clause does not prevent you from retrieving all the qualifying rows. However, if you use OPTIMIZE FOR *n* ROWS, the total elapsed time to retrieve all the qualifying rows might be significantly greater than if DB2 had optimized for the entire result set.

***When OPTIMIZE FOR n ROWS is effective:*** OPTIMIZE FOR *n* ROWS is effective only on queries that can be performed incrementally. If the query causes DB2 to gather the whole result set before returning the first row, DB2 ignores the OPTIMIZE FOR *n* ROWS clause, as in the following situations:

- The query uses SELECT DISTINCT or a set function distinct, such as COUNT(DISTINCT C1).
- Either GROUP BY or ORDER BY is used, and there is no index that can give the ordering necessary.
- There is a column function and no GROUP BY clause.
- The query uses UNION.

***Example:*** Suppose you query the employee table regularly to determine the employees with the highest salaries. You might use a query like this:

```
SELECT LASTNAME, FIRSTNAME, EMPNO, SALARY
  FROM EMP
  ORDER BY SALARY DESC;
```

An index is defined on column EMPNO, so employee records are ordered by EMPNO. If you have also defined a descending index on column SALARY, that index is likely to be very poorly clustered. To avoid many random, synchronous I/O operations, DB2 would most likely use a table space scan, then sort the rows on SALARY. This technique can cause a delay before the first qualifying rows can be returned to the application. If you add the OPTIMIZE FOR *n* ROWS clause to the statement, as shown below:

```
SELECT LASTNAME,FIRSTNAME,EMPNO,SALARY
  FROM EMP
  ORDER BY SALARY DESC
  OPTIMIZE FOR 20 ROWS;
```

DB2 would most likely use the SALARY index directly because you have indicated that you will probably retrieve the salaries of only the 20 most highly paid employees. This choice avoids a costly sort operation.

***Effects of using OPTIMIZE FOR n ROWS:***

- The join method could change. Nested loop join is the most likely choice, because it has low overhead cost and appears to be more efficient if you want to retrieve only one row.
- An index that matches the ORDER BY clause is more likely to be picked. This is because no sort would be needed for the ORDER BY.
- List prefetch is less likely to be picked.
- Sequential prefetch is less likely to be requested by DB2 because it infers that you only want to see a small number of rows.
- In a join query, the table with the columns in the ORDER BY clause is likely to be picked as the outer table if there is an index on that outer table that gives the ordering needed for the ORDER BY clause.

***Recommendation:*** For a local query, specify OPTIMIZE FOR *n* ROWS only in applications that frequently fetch only a small percentage of the total rows in a query result set. For example, an application might read only enough rows to fill the end user's terminal screen. In cases like this, the application might read the remaining part of the query result set only rarely. For an application like this, OPTIMIZE FOR *n* ROWS can result in better performance by causing DB2 to favor SQL access paths that deliver the first n rows as fast as possible.

When you specify OPTIMIZE FOR *n* ROWS for a remote query, a small value of *n* can help limit the number of rows that flow across the network on any given transmission.

You can improve the performance for receiving a large result set through a remote query by specifying a large value of *n* in OPTIMIZE FOR *n* ROWS. When you specify a large value, DB2 attempts to send the *n* rows in multiple transmissions. For better performance when retrieving a large result set, in addition to specifying OPTIMIZE FOR *n* ROWS with a large value of *n* in your query, do not execute other SQL statements until the entire result set for the query is processed. If retrieval of data for several queries overlaps, DB2 might need to buffer result set data in the DDF address space. See “Block fetching result sets” on page 859 for more information.

For local or remote queries, to influence the access path most, specify OPTIMIZE for 1 ROW. This value does not have a detrimental effect on distributed queries.

| 
## Fetching a limited number of rows: FETCH FIRST n ROWS ONLY

| In some applications, you execute queries that can return a large number of rows,
| but you need only a small subset of those rows. Retrieving the entire result table
| from the query can be inefficient. You can specify the FETCH FIRST *n* ROWS

ONLY clause in a SELECT statement to limit the number of rows in the result table of a query to *n* rows. In addition, for a distributed query that uses DRDA access, FETCH FIRST *n* ROWS ONLY, DB2 prefetches only *n* rows.

***Example:*** Suppose that you write an application that requires information on only the 20 employees with the highest salaries. To return only the rows of the employee table for those 20 employees, you can write a query like this:

```
SELECT LASTNAME, FIRSTNAME, EMPNO, SALARY
  FROM EMP
  ORDER BY SALARY DESC
  FETCH FIRST 20 ROWS ONLY;
```

***Interaction between OPTIMIZE FOR n ROWS and FETCH FIRST n ROWS ONLY:*** In general, if you specify FETCH FIRST *n* ROWS ONLY but not OPTIMIZE FOR *n* ROWS in a SELECT statement, DB2 optimizes the query as if you had specified OPTIMIZE FOR *n* ROWS. If you specify the OPTIMIZE FOR *n* ROWS and the FETCH FIRST *m* ROWS clauses, and *n*<*m*, DB2 optimizes the query for *n* rows. If *m*<*n*, DB2 optimizes for *m* rows.

## Reducing the number of matching columns

Discourage the use of a poorer performing index by reducing the index's matching predicate on its leading column. Consider the example in Figure 93 on page 751, where the index that DB2 picks is less than optimal.

DB2 picks IX2 to access the data, but IX1 would be roughly 10 times quicker. The problem is that 50% of all parts from center number 3 are still in Center 3; they have not moved. Assume that there are no statistics on the correlated columns in catalog table SYSCOLDIST. Therefore, DB2 assumes that the parts from center number 3 are evenly distributed among the 50 centers.

You can get the desired access path by changing the query. To discourage the use of IX2 for this particular query, you can change the third predicate to be nonindexable.

```
SELECT * FROM PART_HISTORY
WHERE
    PART_TYPE = 'BB'
AND W_FROM = 3
AND (W_NOW = 3 + 0)      <-- PREDICATE IS MADE NONINDEXABLE
```

Now index I2 is not picked, because it has only one match column. The preferred index, I1, is picked. The third predicate is a nonindexable predicate, so an index is not used for the compound predicate.

There are many ways to make a predicate nonindexable. The recommended way is to make the add 0 to a predicate that evaluates to a numeric value or concatenate a predicate that evaluates to a character value with an empty string.

| Indexable | Nonindexable |
|---|---|
| `T1.C3=T2.C4` | `(T1.C3=T2.C4 CONCAT '')` |
| `T1.C1=5` | `T1.C1=5+0` |

These techniques do not affect the result of the query and cause only a small amount of overhead.

The preferred technique for improving the access path when a table has correlated columns is to generate catalog statistics on the correlated columns. You can do that either by running RUNSTATS or by updating catalog table SYSCOLDIST or SYSCOLDISTSTATS manually.

```
CREATE TABLE PART_HISTORY (
    PART_TYPE  CHAR(2),       IDENTIFIES THE PART TYPE
    PART_SUFFIX CHAR(10),     IDENTIFIES THE PART
    W_NOW      INTEGER,       TELLS WHERE THE PART IS
    W_FROM     INTEGER,       TELLS WHERE THE PART CAME FROM
    DEVIATIONS INTEGER,       TELLS IF ANYTHING SPECIAL WITH THIS PART
    COMMENTS        CHAR(254),
    DESCRIPTION     CHAR(254),
    DATE1           DATE,
    DATE2           DATE,
    DATE3           DATE);

  CREATE UNIQUE INDEX IX1 ON PART_HISTORY
    (PART_TYPE,PART_SUFFIX,W_FROM,W_NOW);
  CREATE UNIQUE INDEX IX2 ON PART_HISTORY
    (W_FROM,W_NOW,DATE1);
  +------------------------------------------------------------------------------+
  | Table statistics              |        Index statistics   IX1         IX2    |
  |-------------------------------+----------------------------------------------|
  | CARDF             100,000     |       FIRSTKEYCARDF       1000          50    |
  | NPAGES             10,000     |       FULLKEYCARDF     100,000     100,000    |
  |                               |       CLUSTERRATIO         99%         99%    |
  |                               |       NLEAF              3000        2000    |
  |                               |       NLEVELS               3           3    |
  |------------------------------------------------------------------------------|
  |                column      cardinality    HIGH2KEY     LOW2KEY               |
  |                --------    -----------    --------     -------               |
  |                Part_type    1000          'ZZ'        'AA'                   |
  |                w_now          50           1000         1                    |
  |                w_from         50           1000         1                    |
  +------------------------------------------------------------------------------+

  Q1:
  SELECT * FROM PART_HISTORY   --  SELECT ALL PARTS
  WHERE PART_TYPE = 'BB'    P1 --  THAT ARE 'BB' TYPES
    AND W_FROM = 3          P2 --  THAT WERE MADE IN CENTER 3
    AND W_NOW = 3           P3 --  AND ARE STILL IN CENTER 3

  +------------------------------------------------------------------------------+
  |   Filter factor of these predicates.                                         |
  |    P1 = 1/1000= .001                                                         |
  |    P2 = 1/50  = .02                                                          |
  |    P3 = 1/50  = .02                                                          |
  |------------------------------------------------------------------------------|
  | ESTIMATED VALUES                   |     WHAT REALLY HAPPENS                 |
  |                    filter    data  |                        filter   data    |
  | index  matchcols   factor    rows  |     index  matchcols   factor   rows    |
  |  ix2     2        .02*.02      40  |      ix2     2        .02*.50    1000    |
  |  ix1     1        .001        100  |      ix1     1        .001        100    |
  +------------------------------------------------------------------------------+
```

*Figure 93. Reducing the number of MATCHCOLS*

## Adding extra local predicates

Adding local predicates on columns that have no other predicates generally has the following effect on join queries.

1. The table with the extra predicates is more likely to be picked as the outer table. That is because DB2 estimates that fewer rows qualify from the table if there are more predicates. It is generally more efficient to have the table with the fewest qualifying rows as the outer table.

2. The join method is more likely to be nested loop join. This is because nested loop join is more efficient for small amounts of data, and more predicates make DB2 estimate that less data is to be retrieved.

The proper type of predicate to add is `WHERE TX.CX=TX.CX`.

This does not change the result of the query. It is valid for a column of any data type, and causes a minimal amount of overhead. However, DB2 uses only the best filter factor for any particular column. So, if TX.CX already has another equal predicate on it, adding this extra predicate has no effect. You should add the extra local predicate to a column that is not involved in a predicate already. If index-only access is possible for a table, it is generally not a good idea to add a predicate that would prevent index-only access.

## Creating indexes for efficient star schemas

A *star schema* is a database design that, in its simplest form, consists of a large table called a *fact table,* and two or more smaller tables, called *dimension tables.* More complex star schemas can be created by breaking one or more of the dimension tables into multiple tables.

To access the data in a star schema, you write SELECT statements that include join operations between the fact table and the dimension tables, but no join operations between dimension tables.

DB2 uses a special join type called a *star join* if the conditions that are described in “Star schema (star join)” on page 820 are true.

You can improve the performance of star joins by your use of indexes. This section gives suggestions for choosing indexes might improve star join performance.

### Recommendations for creating indexes for star schemas

Follow these recommendations to improve performance of queries that are processed using the star join technique:

- Define a multi-column index on all key columns of the fact table. Key columns are fact table columns that have corresponding dimension tables.
- If you do not have information about the way that your data is used, first try a multi-column index on the fact table that is based on the correlation of the data. Put less highly correlated columns later in the index key than more highly correlated columns. See “Determining the order of columns in an index for a star schema” on page 753 for information on deriving an index that follows this recommendation.
- As the correlation of columns in the fact table changes, reevaluate the index to determine if columns in the index should be reordered.
- Define indexes on dimension tables to improve access to those tables.
- When you have executed a number of queries and have more information about the way that the data is used, follow these recommendations:
  - Put more selective columns at the beginning of the index.
  - If a number of queries do not reference a dimension, put the column that corresponds to that dimension at the end of the index.

    When there are multiple multi-column indexes on the fact table, and none of those indexes contain all key columns, DB2 evaluates all of the indexes and uses the index that best exploits star join.

### Determining the order of columns in an index for a star schema

You can use the following method to determine the order of columns in a multi-column index. The description of the method uses the following terminology:

**F** A fact table.

**D1...D*n***
Dimension tables.

**C1...C*n***
Key columns in the fact table. C1 is joined to dimension D1, C2 is joined to dimension D2, and so on.

**cardD1...cardD*n***
Cardinality of columns C1...C*n* in dimension tables D1...D*n*.

**cardC1...cardC*n***
Cardinality of key columns C1...C*n* in fact table F.

**cardC*ij***
Cardinality of pairs of column values from key columns C*i* and C*j* in fact table F.

**cardC*ijk***
Cardinality of triplets of column values from key columns C*i*, C*j*, and C*k* in fact table F.

**Density**
A measure of the correlation of key columns in the fact table. The density is calculated as follows:

**For a single column**
cardC*i*/cardD*i*

**For pairs of columns**
cardC*ij*/(cardD*i**cardD*j*)

**For triplets of columns**
cardC*ijk*/(cardD*i**cardD*j**cardD*k*)

**S** The current set of columns whose order in the index is not yet determined.

**S-{C*m*}**
The current set of columns, excluding column C*m*

Follow these steps to derive a fact table index for a star join that joins *n* columns of fact table F to *n* dimension tables D1 through D*n*:

1. Define the set of columns whose index key order is to be determined as the *n* columns of fact table F that correspond to dimension tables. That is, S={C1,...C*n*} and L=*n*.
2. Calculate the density of all sets of L-1 columns in S.
3. Find the lowest density. Determine which column is not in the set of columns with the lowest density. That is, find column C*m* in S, such that for every C*i* in S, density(S-{C*m*})<density(S-{C*i*}).
4. Make C*m* the Lth column of the index.
5. Remove C*m* from S.
6. Decrement L by 1.
7. Repeat steps 2 through 6 *n*-2 times. The remaining column after iteration *n*-2 is the first column of the index.

***Example of determining column order for a fact table index:*** Suppose that a star schema has three dimension tables with the following cardinalities:

```
cardD1=2000
cardD2=500
cardD3=100
```

Now suppose that the cardinalities of single columns and pairs of columns in the fact table are:

```
cardC1=2000
cardC2=433
cardC3=100
cardC12=625000
cardC13=196000
cardC23=994
```

Determine the best multi-column index for this star schema.

Step 1: Calculate the density of all pairs of columns in the fact table:

```
density(C1,C2)=625000⁄(2000*500)=0.625
density(C1,C3)=196000⁄(2000*100)=0.98
density(C2,C3)=994⁄(500*100)=0.01988
```

Step 2: Find the pair of columns with the lowest density. That pair is (C2,C3). Determine which column of the fact table is not in that pair. That column is C1.

Step 3: Make column C1 the third column of the index.

Step 4: Repeat steps 1 through 3 to determine the second and first columns of the index key:

```
density(C2)=433⁄500=0.866
density(C3)=100⁄100=1.0
```

The column with the lowest density is C2. Therefore, C3 is the second column of the index. The remaining column, C2, is the first column of the index. That is, the best order for the multi-column index is C2, C3, C1.

## Rearranging the order of tables in a FROM clause

The order of tables or views in the FROM CLAUSE can affect the access path. If your query performs poorly, it could be because the join sequence is inefficient. You can determine the join sequence within a query block from the PLANNO column in the PLAN_TABLE. For information on using the PLAN_TABLE, see “Chapter 33. Using EXPLAIN to improve SQL performance” on page 789. If you think that the join sequence is inefficient, try rearranging the order of the tables and views in the FROM clause to match a join sequence that might perform better. Rearranging the columns might cause DB2 to select the better join sequence.

## Updating catalog statistics

If you have the proper authority, it is possible to influence access path selection by using an SQL UPDATE or INSERT statement to change statistical values in the DB2 catalog. However, this is not generally recommended except as a last resort. While updating catalog statistics can help a certain query, other queries can be affected adversely. Also, the UPDATE statements must be repeated after RUNSTATS resets the catalog values. You should be very careful if you attempt to update statistics. For a list of catalog statistics that you can update, see Table 109 on page 766.

The example shown in Figure 93 on page 751, involving this query:

```
SELECT * FROM PART_HISTORY   --  SELECT ALL PARTS
WHERE PART_TYPE = 'BB'    P1 --  THAT ARE 'BB' TYPES
  AND W_FROM = 3          P2 --  THAT WERE MADE IN CENTER 3
  AND W_NOW = 3           P3 --  AND ARE STILL IN CENTER 3
```

is a problem with data correlation. DB2 does not know that 50% of the parts that were made in Center 3 are still in Center 3. It was circumvented by making a predicate nonindexable. But suppose there are hundreds of users writing queries similar to that query. It would not be possible to have all users change their queries. In this type of situation, the best solution is to change the catalog statistics.

For the query in Figure 93 on page 751, where the correlated columns are concatenated key columns of an index, you can update the catalog statistics in one of two ways:

- Run the RUNSTATS utility, and request statistics on the correlated columns W_FROM and W_NOW. This is the preferred method. See "Gathering monitor and update statistics" on page 775 and Part 2 of *DB2 Utility Guide and Reference*for more information.
- Update the catalog statistics manually.

***Updating the catalog to adjust for correlated columns:*** One catalog table you can update is SYSIBM.SYSCOLDIST, which gives information about the first key column or concatenated columns of an index key. Assume that because columns W_NOW and W_FROM are correlated, there are only 100 distinct values for the combination of the two columns, rather than 2500 (50 for W_FROM * 50 for W_NOW). Insert a row like this to indicate the new cardinality:

```
INSERT INTO SYSIBM.SYSCOLDIST
       (FREQUENCY, FREQUENCYF, IBMREQD,
        TBOWNER, TBNAME, NAME, COLVALUE,
        TYPE, CARDF, COLGROUPCOLNO, NUMCOLUMNS)
 VALUES(0, -1, 'N',
        'USRT001','PART_HISTORY','W_FROM',' ',
        'C',100,X'00040003',2);
```

Because W_FROM and W_NOW are concatenated key columns of an index, you can also put this information in SYSCOLDIST using the RUNSTATS utility. See *DB2 Utility Guide and Reference* for more information.

You can also tell DB2 about the frequency of a certain combination of column values by updating SYSIBM.SYSCOLDIST. For example, you can indicate that 1% of the rows in PART_HISTORY contain the values 3 for W_FROM and 3 for W_NOW by inserting this row into SYSCOLDIST:

```
INSERT INTO SYSIBM.SYSCOLDIST
       (FREQUENCY, FREQUENCYF, STATSTIME, IBMREQD,
        TBOWNER, TBNAME, NAME, COLVALUE,
        TYPE, CARDF, COLGROUPCOLNO, NUMCOLUMNS)
 VALUES(0, .0100, '1996-12-01-12.00.00.000000','N',
        'USRT001','PART_HISTORY','W_FROM',X'00800000030080000003',
        'F',-1,X'00040003',2);
```

***Updating the catalog for joins with table functions:*** Updating catalog statistics **might cause extreme performance problems** if the statistics are not updated correctly. Monitor performance, and be prepared to reset the statistics to their original values if performance problems arise.

## Using a subsystem parameter

This section describes subsystem parameters that influence access path selection. To set these subsystem parameters, you modify and run installation job DSNTIJUZ, then restart DB2. See Part 2 of *DB2 Installation Guide* for detailed information on how to set subsystem parameters.

### Using a subsystem parameter to favor matching index access

DB2 often does a table space scan or nonmatching index scan when the data access statistics indicate that a table is small, even though matching index access is possible. This is a problem if the table is small or empty when statistics are collected, but the table is large when it is queried. In that case, the statistics are not accurate and can lead DB2 to pick an inefficient access path.

The best solution to the problem is to run RUNSTATS again after the table is populated. However, if it is not possible to do that, you can use subsystem parameter NPGTHRSH to cause DB2 to favor matching index access over a table space scan and over nonmatching index access.

NPGTHRSH is in macro DSN6SPRM. The value of NPGTHRSH is an integer that indicates the tables for which DB2 favors matching index access. Values of NPGTHRSH and their meanings are:

**−1** DB2 favors matching index access for all tables.

**0** DB2 selects the access path based on cost, and no tables qualify for special handling. This is the default.

*n*>=1 If data access statistics have been collected for all tables, DB2 favors matching index access for tables for which the total number of pages on which rows of the table appear (NPAGES) is less than *n*.

If data access statistics have not been collected for some tables (NPAGES=-1 for those tables), DB2 favors matching index access for tables for which NPAGES=-1 or NPAGES<*n*.

***Recommendation:*** Before you use NPGTHRSH, be aware that in some cases, matching index access can be more costly than a table space scan or nonmatching index access. Specify a small value for NPGTHRSH (10 or less). That limits the number of tables for which DB2 favors matching index access.

### Using a subsystem parameter to control outer join processing

Subsystem parameter OJPERFEH can improve outer join processing. In particular, when the value of OJPERFEH is YES, DB2 takes the following actions, which can improve outer join processing in most cases:

- Does not merge table expressions or views if the parent query block of a table expression or view contains an outer join, and the merge would cause a column in a predicate to become an expression.
- Does not attempt to reduce work file usage for outer joins.
- Uses transitive closure for the ON predicates in outer joins.

However, these actions might not improve performance for some outer joins.

***Recommendation:*** If the performance of queries that contain outer joins is not adequate, set OJPERFEH to NO, restart DB2, and rerun those queries.

## Giving optimization hints to DB2

This section describes how experienced programmer analysts can tell DB2 how to process a query. You do this by giving DB2 *hints*. The process of giving hints to DB2 is relatively simple but determining what those hints should be is not. Always test and analyze the results of any query that uses optimization hints.

Giving optimization hints to DB2 is useful in the following situations:

- You want to ensure consistency of response times across rebinds and across release migrations. When a plan or package is rebound, the access path is reformulated. If the database or application has changed, or if DB2 has new function that causes it to choose a different access path, it is handy to have the ability to use an old access path if the new one does not perform as well.

  For this reason, it is a good idea to save the access paths of your critical queries before migrating to a new release of DB2.
- You want to temporarily bypass the access path chosen by DB2.

This section describes the following tasks:

- “Planning to use optimization hints”
- “Enabling optimization hints for the subsystem”
- “Scenario: Preventing a change at rebind”
- “Scenario: Modifying an existing access path” on page 759
- “Reasons to use the QUERYNO clause” on page 760
- “How DB2 validates the hint” on page 761

### Planning to use optimization hints

Before you can give hints to DB2, make sure your PLAN_TABLE is of the correct format. The steps are:

1. Migrate your existing authid.PLAN_TABLE to the 49-column format described in Figure 99 on page 791.
2. For best performance, create an ascending index on the following columns of PLAN_TABLE:
   - QUERYNO
   - APPLNAME
   - PROGNAME
   - VERSION
   - COLLID
   - OPTHINT

   The DB2 sample library, in member DSNTESC, contains an appropriate CREATE INDEX statement that you can modify.

### Enabling optimization hints for the subsystem

On the subsystem where the application is bound or where dynamic queries are prepared, specify YES in the OPTIMIZATION HINTS field of installation panel DSNTIP4. If you specify NO, DB2 ignores any hints.

### Scenario: Preventing a change at rebind

The following scenario assumes that DB2 is using an access path that you like for a particular query and that there is currently a row in the PLAN_TABLE for that desired access path. By making that access path a hint, DB2 will use your hint when you rebind or migrate to a new release.

1. Determine the query number that currently exists for that query in the PLAN_TABLE. To ensure that the query number is always correlated with that query in the application, modify the statement in the application to use the

QUERYNO clause. (If you want to use some kind of numbering convention for queries that use access path hints, you can change the query number in PLAN_TABLE. The important thing is to have the query in the application have a query number that is unique for that application and that matches the QUERYNO value in the PLAN_TABLE.)

Here is an example of the QUERYNO clause:

```
SELECT * FROM T1
  WHERE C1 = 10 AND
  C2 BETWEEN 10 AND 20 AND
  C3 NOT LIKE 'A%'
  QUERYNO 100;
```

For more information about reasons to use the QUERYNO clause, see “Reasons to use the QUERYNO clause” on page 760.

2. Make the PLAN_TABLE rows for that query (QUERYNO=100) into a hint by updating the OPTHINT column with the name you want to call the hint. In this case, the name is OLDPATH:

```
UPDATE PLAN_TABLE
  SET OPTHINT = 'OLDPATH' WHERE
  QUERYNO = 100 AND
   APPLNAME = ' ' AND
   PROGNAME = 'DSNTEP2' AND
   VERSION = '' AND
   COLLID = 'DSNTEP2';
```

3. Tell DB2 to use the hint, and indicate in the PLAN_TABLE that DB2 used the hint.
   - For dynamic SQL statements in the program, follow these steps:
     a. Execute the SET CURRENT OPTIMIZATION HINT statement in the program to tell DB2 to use OLDPATH. For example:

        ```
        SET CURRENT OPTIMIZATION HINT = 'OLDPATH';
        ```

        If you do not explicitly set the CURRENT OPTIMIZATION HINT special register, the value that you specify for the bind option OPTHINT is used.

        If you execute the SET CURRENT OPTIMIZATION HINT statement statically, rebind the plan or package to pick up the SET CURRENT OPTIMIZATION HINT statement.
     b. Execute the EXPLAIN statement on the SQL statements for which you have instructed DB2 to use OLDPATH. This step adds rows to the PLAN_TABLE for those statements. The rows contain a value of OLDPATH in the HINT_USED column.

        If DB2 uses the hint you provided, it returns SQLCODE +394 from the PREPARE of the EXPLAIN statement and from the PREPARE of SQL statements that use the hint. If your hints are invalid, DB2 issues SQLCODE +395.
   - For static SQL statements in the program, rebind the plan or package that contains the statements. Specify bind options EXPLAIN(YES) and OPTHINT('OLDPATH') to add rows for those statements in the PLAN_TABLE that contain a value of OLDPATH in the HINT_USED column.

     If DB2 uses the hint you provided, it returns SQLCODE +394 from the rebind. If your hints are invalid, DB2 issues SQLCODE +395.
4. Select from PLAN_TABLE to see what was used:

```
SELECT *
  FROM PLAN_TABLE
   WHERE QUERYNO = 100
  ORDER BY TIMESTAMP, QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

The PLAN_TABLE in Table 107 shows the OLDPATH hint, indicated by a value in OPTHINT and it also shows that DB2 used that hint, indicated by OLDPATH in the HINT_USED column.

*Table 107. PLAN_TABLE that shows that the OLDPATH optimization hint is used.*

| QUERYNO | METHOD | TNAME | OPTHINT | HINT_USED |
|---|---|---|---|---|
| 100 | 0 | EMP | OLDPATH | |
| 100 | 4 | EMPPROJACT | OLDPATH | |
| 100 | 3 | | OLDPATH | |
| 100 | 0 | EMP | | OLDPATH |
| 100 | 4 | EMPPROJECT | | OLDPATH |
| 100 | 3 | | | OLDPATH |

## Scenario: Modifying an existing access path

The following scenario assumes that DB2 is using a hybrid join where you know it can perform better with a sort merge join. The example assumes that the query is dynamic.

1. Put the old access path in the PLAN_TABLE and associate it with a query number.

   ```
   EXPLAIN ALL SET QUERYNO=200 FOR
   SELECT X.ACTNO, X.PROJNO, X.EMPNO, Y.JOB, Y.EDLEVEL
     FROM DSN8610.EMPPROJACT X, DSN8610.EMP Y
    WHERE X.EMPNO = Y.EMPNO
      AND X.EMPTIME > 0.5
      AND (Y.JOB = 'DESIGNER' OR Y.EDLEVEL >= 12)
    ORDER BY X.ACTNO, X.PROJNO;
   ```

2. Make the PLAN_TABLE rows into a hint by updating the OPTHINT column with the name you want to call the hint. In this case, the name is NOHYB:

   ```
   UPDATE PLAN_TABLE
     SET OPTHINT = 'NOHYB' WHERE
     QUERYNO = 200 AND
      APPLNAME = ' ' AND
      PROGNAME = 'DSNTEP2' AND
      VERSION = '' AND
      COLLID = 'DSNTEP2';
   ```

3. Change the access path so that merge scan join is used rather than hybrid join:

   ```
   UPDATE PLAN_TABLE
     SET METHOD = 2 WHERE
     QUERYNO = 200 AND
     APPLNAME = ' ' AND
     PROGNAME = 'DSNTEP2' AND
     VERSION = '' AND
     COLLID = 'DSNTEP2' AND
     OPTHINT = 'NOHYB' AND
     METHOD = 4;
   ```

4. Tell DB2 to look for the NOHYB hint for this query:

   ```
   SET CURRENT OPTIMIZATION HINT = 'NOHYB';
   ```

5. Explain the query again to check for the results:

```
EXPLAIN ALL SET QUERYNO=200 FOR
SELECT X.ACTNO, X.PROJNO, X.EMPNO, Y.JOB, Y.EDLEVEL
  FROM DSN8610.EMPPROJACT X, DSN8610.EMP Y
 WHERE X.EMPNO = Y.EMPNO
   AND X.EMPTIME > 0.5
   AND (Y.JOB = 'DESIGNER' OR Y.EDLEVEL >= 12)
 ORDER BY X.ACTNO, X.PROJNO;
```

6. Select from the PLAN_TABLE to verify the results:

```
SELECT *
  FROM PLAN_TABLE
   WHERE QUERYNO = 200
  ORDER BY TIMESTAMP, QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

The PLAN_TABLE in Table 108 shows the NOHYB hint, indicated by a value in OPTHINT and it also shows that DB2 used that hint, indicated by NOHYB in the HINT_USED column.

*Table 108. PLAN_TABLE that shows that the NOHYB optimization hint is used.*

| QUERYNO | METHOD | TNAME | OPTHINT | HINT_USED |
|---|---|---|---|---|
| 200 | 0 | EMP | NOHYB | |
| 200 | 2 | EMPPROJACT | NOHYB | |
| 200 | 3 | | NOHYB | |
| 200 | 0 | EMP | | NOHYB |
| 200 | 2 | EMPPROJECT | | NOHYB |
| 200 | 3 | | | NOHYB |

7. Analyze the performance of the statement to see if it is acceptable.

## Reasons to use the QUERYNO clause

You do not need to assign a query number to use optimization hints. If you don’t assign a query number, DB2 uses the statement number. However, assigning a query number is especially useful in the following cases:

- For dynamic statements

  The query number for dynamic applications is the statement number in the application *where the prepare occurs*. For some applications, such as DSNTEP2, the same statement in the application prepares each dynamic statement, resulting in the same query number for each dynamic statement. Assigning a query number to each statement that uses optimization hints eliminates ambiguity as to which rows in the PLAN_TABLE are associated with each query.

- For static statements

  If you change an application that has static statements, the statement number might change, causing rows in the PLAN_TABLE to be out of sync with the modified application. Statements that use the QUERYNO clause are not dependent on the statement number. You can move those statements around without affecting the relationship between rows in the PLAN_TABLE and the statements that use those rows in the application.

## How DB2 locates the PLAN_TABLE rows for a hint

DB2 uses the QUERYNO, APPLNAME, PROGNAME, VERSION, COLLID, and OPTHINT columns of the PLAN_TABLE to determine the rows to use for a hint. For a PLAN_TABLE row, the QUERYNO, APPLNAME, PROGNAME, VERSION, and COLLID values must match the corresponding values for an SQL statement before the SQL statement can use that row. In addition, the OPTHINT value for that row must match the value in the CURRENT OPTIMIZATION HINT special register if the

SQL statement is executed dynamically. If the SQL statement is executed statically, the OPTHINT value for the row must match the value of bind option OPTHINT for the package or plan that contains the SQL statement. If no PLAN_TABLE rows meet these conditions, DB2 determines the access path for the SQL statement without using hints.

## How DB2 validates the hint

When you specify an optimization hint (OPTHINT='*hint-id*')), DB2 validates the information in the PLAN_TABLE to ensure that you chose a valid access path. If the access path you specify has major problems, DB2 invalidates all hints for that query block. In that event, DB2 determines the access path as it normally does.

Here are the valid values:

**Column**
: **Correct Values or Other Explanation**

**METHOD**
: Must be 0, 1, 2, 3, or 4. Any other value invalidates the hints. See "Interpreting access to two or more tables (join)" on page 812 for more information about join methods.

**CREATOR and TNAME**
: Must be specified and must name a table, materialized view, materialized nested table expression. Blank if method is 3. If a table is named that does not exist or is not involved in the query, then the hints are invalid.

**CORRELATION_NAME:**
: Required only if CREATOR and TNAME do not uniquely identify the table. If you enter a value, the value must match the correlation name specified in the query or in the CREATE VIEW statement.

**TABNO**
: Required only if CREATOR, TNAME, and CORRELATION_NAME do not uniquely identify the table. This situation might occur when the same table is used in multiple views (with the same CORRELATION_NAME).

  This field is ignored if it is not needed.

**ACCESSTYPE**
: Must contain R, I, I1, N, or M. Any other value invalidates the hints.

  Values of I, I1, and N all mean single index access. DB2 determines which of the 3 values to use based on the index specified in ACCESSNAME.

  M indicates multiple index access. DB2 uses only the first row in the authid.PLAN_TABLE for multiple index access (MIXOPSEQ=0). The choice of indexes, and the AND and OR operations involved, is determined by DB2. If multiple index access isn't possible, then the hints are invalidated.

  See "Is access through an index? (ACCESSTYPE is I, I1, N or MX)" on page 799 and "Is access through more than one index? (ACCESSTYPE=M)" on page 799 for more information.

**ACCESSCREATOR and ACCESSNAME**
: Ignored if ACCESSTYPE is R or M. If ACCESSTYPE is I, I1, or N, then these fields must identify an index on the specified table.

  If the index doesn't exist, or if the index is defined on a different table, then the hints are invalid. Also, if the specified index can't be used, then the hints are invalid.

**SORTN_JOIN and SORTC_JOIN**

Must be Y, N or blank. Any other value invalidates the hints.

This value determines if DB2 should sort the new (SORTN_JOIN) or composite (SORTC_JOIN) table. This value is ignored if the specified join method, join sequence, access type and access name dictate whether a sort of the new or composite tables is required.

See “Are sorts performed?” on page 804 for more information.

**PREFETCH**

Must be S, L or blank. Any other value invalidates the hints.

This value determines whether DB2 should use sequential prefetch (S), list prefetch (L), or no prefetch (blank). (A blank does not prevent sequential detection at run time.) This value is ignored if the specified access type and access name dictates the type of prefetch required.

See “What kind of prefetching is done? (PREFETCH = L, S, or blank)” on page 803 for more information.

**PAGE_RANGE**

Must be Y, N or blank. Any other value invalidates the hints. See “Was a scan limited to certain partitions? (PAGE_RANGE=Y)” on page 803 for more information.

**PARALLELISM_MODE**

This value is used only if it is possible to run the query in parallel; that is, the SET CURRENT DEGREE special register contains ANY, or the plan or package was bound with DEGREE(ANY).

If parallelism is possible, this value must be I, C, X or null. All of the restrictions involving parallelism still apply when using access path hints. If the specified mode cannot be performed, the hints are either be invalidated or the mode is modified by the optimizer, possibly resulting in the query being run sequentially. If the value is null then the optimizer determines the mode.

See “Chapter 34. Parallel operations and query performance” on page 841 for more information.

**ACCESS_DEGREE or JOIN_DEGREE**

If PARALLELISM_MODE is specified, use this field to specify the degree of parallelism. If you specify a degree of parallelism, this must a number greater than zero, and DB2 might adjust the parallel degree from what you set here. If you want DB2 to determine the degree, do not enter a value in this field.

If you specify a value for ACCESS_DEGREE or JOIN_DEGREE, you must also specify a corresponding ACCESS_PGROUP_ID and JOIN_PGROUP_ID.

**WHEN_OPTIMIZE**

Must be R, B, or blank. Any other value invalidates the hints.

When a statement in a plan that is bound with REOPT(VARS) qualifies for reoptimization at run time, and you have provided optimization hints for that statement, the value of WHEN_OPTIMIZE determines whether DB2 reoptimizes the statement at run time. If the value of WHEN_OPTIMIZE is blank or B, DB2 uses only the access path that is provided by the optimization hints at bind time. If the value of WHEN_OPTIMIZE is R, DB2 determines the access path at bind time using the optimization hints. At run time, DB2 searches the PLAN_TABLE for hints again, and if hints for the statement are still in the PLAN_TABLE and are still valid, DB2 optimizes the access path using those hints again.

**PRIMARY_ACCESSTYPE**
Must be D or blank. Any other value invalidates the hints.

# Chapter 32. Maintaining statistics in the catalog

Statistics stored in the DB2 catalog help DB2 determine the access paths selected for your SQL statements.

- Table 109 on page 766 lists the tables and columns in the catalog that contain those statistics.
- “Understanding statistics used for access path selection” describes some of those columns in more detail.

You can update some of the values in those columns by executing UPDATE statements, and the DB2 utility RUNSTATS can update most of the values.

- “Setting default statistics for created temporary tables” on page 772 explains how to set default statistical values for created temporary tables.
- “Gathering monitor and update statistics” on page 775 contains some advice about running RUNSTATS.
- “Updating the catalog” on page 777 warns of items to watch for if you make your own updates.
- “Querying the catalog for statistics” on page 779 tells you how to find what values are in place.

In addition to access path selection, statistics have the following other uses:

- “Improving index and table space access” on page 780
- “Modeling your production system” on page 786

***Other considerations for access path selection:*** Two other pieces of information that influence access path selection are not stored in the DB2 catalog: the size of the buffer pool and the model type of the central processor (CP).

Access path selection uses buffer pool statistics for several calculations. One is the estimation of the maximum amount of the RID storage pool that can be used. Access path selection also considers the central processor model. These two factors can change your queries’ access paths from one system to another, even if all the catalog statistics are identical. You should keep this in mind when migrating from a test system to a production system, or when modeling a new application.

Mixed central processor models in a data sharing group can also affect access path selection. For more information on data sharing, see *DB2 Data Sharing: Planning and Administration*.

## Understanding statistics used for access path selection

Table 109 lists the statistics in the DB2 catalog that are used for access path selection, the values that trigger the use of a default value, and the corresponding defaults.

Information in the catalog tables SYSTABLES and SYSTABLESPACE tells how much data is in your table and how many pages hold data. Information in SYSINDEXES lets you compare the available indexes on a table to determine which one is the most efficient for a query. SYSCOLUMNS and SYSCOLDIST provide information to estimate filter factors for predicates.

*Table 109. Catalog data used for access path selection or collected by RUNSTATS. Some Version 6 columns are no longer used in Version 7 and are not shown here. They are updated by RUNSTATS but are only used in case of fallback.*

| Column name | Set by RUNSTATS? | User can update? | Used for access paths? [1] | Description |
|---|---|---|---|---|
| **In every table updated by RUNSTATS:** | | | | |
| STATSTIME | Yes | Yes | No | If updated most recently by RUNSTATS, the date and time of that update, not updatable in SYSINDEXPART and SYSTABLEPART. Used for access path selection for SYSCOLDIST if duplicate column values exist for the same column (by user insertion). |
| **SYSIBM.SYSCOLDIST** | | | | |
| CARDF | Yes | Yes | Yes | The number of distinct values for the column group, -1 if TYPE is F |
| COLGROUPCOLNO | Yes | Yes | Yes | The set of columns associated with the statistics. Contains an empty string if NUMCOLUMNS = 1. |
| COLVALUE | Yes | Yes | Yes | Frequently occurring value in the key distribution |
| FREQUENCYF | Yes | Yes | Yes | A number which, multiplied by 100, gives the percentage of rows that contain the value of COLVALUE. For example, 1 means 100% of the rows contain the value and .15 indicates that 15% of the rows contain the value. |
| NUMCOLUMNS | Yes | Yes | Yes | The number of columns associated with the statistics. The **default value** is 1. |
| TYPE | Yes | Yes | Yes | The type of statistics gathered, either cardinality (C) or frequent value (F) |
| **SYSIBM.SYSCOLDISTSTATS:** contains statistics by partition | | | | |
| CARDF | Yes | Yes | No | The number of distinct values for the column group, -1 if TYPE is F |
| COLGROUPCOLNO | Yes | Yes | No | The set of columns associated with the statistics |
| COLVALUE | Yes | Yes | No | Frequently occurring value in the key distribution |
| FREQUENCYF | Yes | Yes | No | A number which, multiplied by 100, gives the percentage of rows that contain the value of COLVALUE. For example, 1 means 100% of the rows contain the value and .15 indicates that 15% of the rows contain the value. |
| NUMCOLUMNS | Yes | Yes | No | The number of columns associated with the statistics. The **default value** is 1. |
| TYPE | Yes | Yes | No | The type of statistics gathered, either cardinality (C) or frequent value (F) |
| **SYSIBM.SYSCOLSTATS:** contains statistics by partition | | | | |
| COLCARD | Yes | Yes | No | The number of distinct values in the partition. Do not update this column manually without first updating COLCARDDATA to a value of length 0. |

*Table 109. Catalog data used for access path selection or collected by RUNSTATS (continued). Some Version 6 columns are no longer used in Version 7 and are not shown here. They are updated by RUNSTATS but are only used in case of fallback.*

| Column name | Set by RUNSTATS? | User can update? | Used for access paths? [1] | Description |
|---|---|---|---|---|
| COLCARDDATA | Yes | Yes | No | The internal representation of the estimate of the number of distinct values in the partition. A value appears here only if RUNSTATS TABLESPACE is run on the partition. Otherwise, this column contains a string of length 0, indicating that the actual value is in COLCARD. |
| HIGHKEY | Yes | Yes | No | First 8 bytes of the highest value of the column within the partition. Blank if LOB column. |
| HIGH2KEY | Yes | Yes | No | First 8 bytes of the second highest value of the column within the partition. Blank if LOB column. |
| LOWKEY | Yes | Yes | No | First 8 bytes of the lowest value of the column within the partition. Blank if LOB column. |
| LOW2KEY | Yes | Yes | No | First 8 bytes of the second lowest value of the column within the partition. Blank if LOB column. |
| **SYSIBM.SYSCOLUMNS** | | | | |
| COLCARDF | Yes | Yes | Yes | Estimated number of distinct values in the column, -1 to trigger DB2's use of the **default value** (25) and -2 for the first column of an index of an auxiliary table |
| HIGH2KEY | Yes | Yes | Yes | First 8 bytes of the second highest value in this column. Blank for auxiliary index. |
| LOW2KEY | Yes | Yes | Yes | First 8 bytes of the second lowest value in this column. Blank for auxiliary index. |
| **SYSIBM.SYSINDEXES** | | | | |
| CLUSTERED | Yes | Yes | No | Whether the table is actually clustered by the index. Blank for auxiliary index. |
| CLUSTERING | No | No | Yes | Whether the index was created using CLUSTER |
| CLUSTERRATIOF | Yes | Yes | Yes | A number which, when multiplied by 100, gives the percentage of rows in clustering order. For example, 1 indicates that all rows are in clustering order and .87825 indicates that 87.825% of the rows are in clustering order. For a partitioned index, it is the weighted average of all index partitions in terms of the number of rows in the partition. For an auxiliary index it is -2. If this columns contains the default, 0, DB2 uses the value in CLUSTERRATIO, a percentage, for access path selection. |
| FIRSTKEYCARDF | Yes | Yes | Yes | Number of distinct values of the first key column, or an estimate if updated while collecting statistics on a single partition, -1 to trigger DB2's use of the **default value** (25) |

*Table 109. Catalog data used for access path selection or collected by RUNSTATS (continued). Some Version 6 columns are no longer used in Version 7 and are not shown here. They are updated by RUNSTATS but are only used in case of fallback.*

| Column name | Set by RUNSTATS? | User can update? | Used for access paths? [1] | Description |
|---|---|---|---|---|
| FULLKEYCARDF | Yes | Yes | Yes | Number of distinct values of the full key, -1 to trigger DB2’s use of the **default value** (25) |
| NLEAF | Yes | Yes | Yes | Number of active leaf pages in the index, -1 to trigger DB2’s use of the **default value** (SYSTABLES.CARD/300) |
| NLEVELS | Yes | Yes | Yes | Number of levels in the index tree, -1 to trigger DB2’s use of the **default value** (2) |
| SPACEF | Yes | Yes | No | Kilobytes of disk storage |
| **SYSIBM.SYSINDEXPART:** contains statistics for space utilization | | | | |
| CARDF | Yes | No | No | Number of rows or LOBs referenced by the index or partition |
| DSNUM | Yes | Yes | No | Number of data sets |
| EXTENTS | Yes | Yes | No | Number of data set extents (when there are multiple pieces, the value is for the extents in the last data set) |
| FAROFFPOSF | Yes | No | No | Number of referenced rows far from the optimal position because of an insert into a full page |
| LEAFDIST | Yes | No | No | 100 times the number of pages between successive leaf pages. |
| LEAFFAR | Yes | Yes | No | Number of leaf pages located physically far away from previous leaf pages for successive active leaf pages accessed in an index scan. See “Understanding LEAFNEAR and LEAFFAR” on page 784 for more information. |
| LEAFNEAR | Yes | Yes | No | Number of leaf pages located physically near previous leaf pages for successive active leaf pages. See “Understanding LEAFNEAR and LEAFFAR” on page 784 for more information. |
| LIMITKEY | No | No | Yes | The limit key of the partition in an internal format, 0 if the index is not partitioned |
| NEAROFFPOSF | Yes | No | No | Number of referenced rows near but not at the optimal position because of an insert into a full page |
| PQTY | Yes | No | No | The primary space allocation in 4K blocks for the data set |
| PSEUDO_DEL_ ENTRIES | Yes | Yes | No | Number of pseudo deleted keys |
| SECQTYI | Yes | No | No | Secondary space allocation in units of 4 KB, stored in integer format instead of small integer format supported by SQTY. If a storage group is not used, the value is 0. |
| SPACE | Yes | No | No | The number of kilobytes of space currently allocated for all extents (contains the accumulated space used by all pieces if a page set contains multiple pieces) |

*Table 109. Catalog data used for access path selection or collected by RUNSTATS (continued). Some Version 6 columns are no longer used in Version 7 and are not shown here. They are updated by RUNSTATS but are only used in case of fallback.*

| Column name | Set by RUNSTATS? | User can update? | Used for access paths? [1] | Description |
|---|---|---|---|---|
| SQTY | Yes | No | No | The secondary space allocation in 4K blocks for the data set |
| SPACEF | Yes | Yes | No | Kilobytes of disk storage |
| **SYSIBM.SYSINDEXSTATS:** contains statistics by partition | | | | |
| CLUSTERRATIOF | Yes | Yes | No | A number which, when multiplied by 100, gives the percentage of rows in clustering order. For example, 1 indicates that all rows are in clustering order and .87825 indicates that 87.825% of the rows are in clustering order. |
| FIRSTKEYCARDF | Yes | Yes | No | Number of distinct values of the first key column, or an estimate if updated while collecting statistics on a single partition |
| FULLKEYCARDF | Yes | Yes | No | Number of distinct values of the full key |
| KEYCOUNTF | Yes | Yes | No | Number of rows in the partition, -1 to trigger DB2's use of the value in KEYCOUNT |
| NLEAF | Yes | Yes | No | Number of leaf pages in the index |
| NLEVELS | Yes | Yes | No | Number of levels in the index tree |
| **SYSIBM.SYSLOBSTATS:** contains LOB table space statistics | | | | |
| AVGSIZE | Yes | Yes | No | Average size of a LOB in bytes |
| FREESPACE | Yes | Yes | No | The number of kilobytes of available space in the LOB table space |
| ORGRATIO | Yes | Yes | No | The ratio of organization in the LOB table space. A value of 1 means perfect organization. The more the value exceeds 1, the more disorganized the LOB table space is. |
| **SYSIBM.SYSROUTINES:** Contains statistics for table functions. See "Updating catalog statistics" on page 754 for more information about using these statistics. | | | | |
| CARDINALITY | No | Yes | Yes | The predicted cardinality of a table function, -1 to trigger DB2's use of the **default value** (10 000) |
| INITIAL_INSTS | No | Yes | Yes | Estimated number of instructions executed the first and last time the function is invoked, -1 to trigger DB2's use of the **default value** (40 000) |
| INITIAL_IOS | No | Yes | Yes | Estimated number of IOs performed the first and last time the function is invoked, -1 to trigger DB2's use of the **default value** (0) |
| INSTS_PER_INVOC | No | Yes | Yes | Estimated number of instructions per invocation, -1 to trigger DB2's use of the **default value** (4 000) |
| IOS_PER_INVOC | No | Yes | Yes | Estimated number of IOs per invocation, -1 to trigger DB2's use of the **default value** (0) |
| **SYSIBM.SYSTABLEPART:** contains statistics for space utilization | | | | |

*Table 109. Catalog data used for access path selection or collected by RUNSTATS (continued). Some Version 6 columns are no longer used in Version 7 and are not shown here. They are updated by RUNSTATS but are only used in case of fallback.*

| Column name | Set by RUNSTATS? | User can update? | Used for access paths? [1] | Description |
|---|---|---|---|---|
| CARDF | Yes | No | No | Total number of rows in the table space or partition. For LOB table spaces, the number of LOBs in the table space. |
| DSNUM | Yes | Yes | No | Number of data sets |
| EXTENTS | Yes | Yes | No | Number of data set extents (when there are multiple pieces, the value is for the extents in the last data set) |
| FARINDREF | Yes | No | No | Number of rows relocated far from their original page |
| NEARINDREF | Yes | No | No | Number of rows relocated near their original page |
| PAGESAVE | Yes | No | No | Percentage of pages, times 100, saved in the table space or partition as a result of using data compression |
| PERCACTIVE | Yes | No | No | Percentage of space occupied by active rows, containing actual data from active tables, -2 for LOB table spaces |
| PERCDROP | Yes | No | No | For nonsegmented table spaces, the percentage of space occupied by rows of data from dropped tables; for segmented table spaces, 0 |
| PQTY | Yes | No | No | The primary space allocation in 4K blocks for the data set |
| SECQTYI | Yes | No | No | Secondary space allocation in units of 4 KB, stored in integer format instead of small integer format supported by SQTY. If a storage group is not used, the value is 0. |
| SPACE | Yes | No | No | The number of kilobytes of space currently allocated for all extents (contains the accumulated space used by all pieces if a page set contains multiple pieces) |
| SPACEF | Yes | Yes | No | Kilobytes of disk storage |
| SQTY | Yes | No | No | The secondary space allocation in 4K blocks for the data set |
| **SYSIBM.SYSTABLES:** | | | | |
| AVGROWLEN | Yes | Yes | No | Average row length of the table specified in the table space |
| CARDF | Yes | Yes | Yes | Total number of rows in the table or total number of LOBs in an auxiliary table, -1 to trigger DB2’s use of the **default value** (10 000) |
| EDPROC | No | No | Yes | Nonblank value if an edit exit routine is used |
| NPAGES | Yes | Yes | Yes | Total number of pages on which rows of this table appear, -1 to trigger DB2’s use of the **default value** (CEILING(1 + CARD/20)) |
| NPAGESF | Yes | Yes | Yes | Number of pages used by the table |

*Table 109. Catalog data used for access path selection or collected by RUNSTATS (continued). Some Version 6 columns are no longer used in Version 7 and are not shown here. They are updated by RUNSTATS but are only used in case of fallback.*

| Column name | Set by RUNSTATS? | User can update? | Used for access paths? [1] | Description |
|---|---|---|---|---|
| PCTPAGES | Yes | Yes | No | For nonsegmented table spaces, percentage of total pages of the table space that contain rows of the table; for segmented table spaces, the percentage of total pages in the set of segments assigned to the table that contain rows of the table |
| PCTROWCOMP | Yes | Yes | Yes | Percentage of rows compressed within the total number of active rows in the table |
| \| SPACEF | Yes | Yes | No | Kilobytes of disk storage |
| **SYSIBM.SYSTABLESPACE:** | | | | |
| NACTIVEF | Yes | Yes | Yes | Number of active pages in the table space, the number of pages touched if a cursor is used to scan the entire file, 0 to trigger DB2's use of the value in the NACTIVE column instead. If NACTIVE contains 0, DB2 uses the **default value** (CEILING(1 + CARD/20)). |
| **SYSIBM.SYSTABSTATS:** contains statistics by partition | | | | |
| CARDF | Yes | Yes | Yes | Total number of rows in the partition, -1 to trigger DB2's use of the value in the CARD column. If CARD is -1, DB2 uses a **default value**(10 000) |
| NACTIVE | Yes | Yes | No | Number of active pages in the partition |
| NPAGES | Yes | Yes | Yes | Total number of pages on which rows of the partition appear, -1 to trigger DB2's use of the **default value** (CEILING(1 + CARD/20)) |
| PCTPAGES | Yes | Yes | No | Percentage of total active pages in the partition that contain rows of the table |
| PCTROWCOMP | Yes | Yes | No | Percentage of rows compressed within the total number of active rows in the partition, -1 to trigger DB2's use of the **default value** (0) |

[1] Statistics on LOB-related values are not used for access path selection. The only exceptions are NLEVELS and FIRSTKEYCARDF for auxiliary indexes. SYSCOLDISTSTATS and SYSINDEXSTATS are not used for parallelism access paths. SYSCOLSTATS information (CARD, HIGHKEY, LOWKEY, HIGH2KEY, and LOW2KEY) is used to determine the degree of parallelism.

## Filter factors and catalog statistics

The catalog tables SYSIBM.SYSCOLUMNS and SYSIBM.SYSCOLDIST are the main source of statistics for calculating predicate filter factors. The following columns are particularly important:

- SYSCOLUMNS.COLCARDF indicates whether statistics exist for a column or not. A value of '-1' results in the use of default statistics. A positive value is an estimate of the number of distinct values in the column.

  The value of COLCARDF generated by RUNSTATS TABLESPACE is an estimate determined by a sampling method. If you know a more accurate number for COLCARDF, you can supply it by updating the catalog. If the column is the first column of an index, the value generated by RUNSTATS INDEX is exact.

- Columns in SYSCOLDIST contain statistics about distributions and correlated key values. Specifying the KEYCARD option of RUNSTATS allows you to collect key cardinality statistics between FIRSTKEYCARDF and FULLKEYCARDF (which are collected by default). Specifying the FREQVAL option of RUNSTATS allows you to specify how many key columns to concatenate and how many frequently occurring values to collect. By default, the 10 most frequently occurring values on the first column of each index are collected. For more information, see Part 2 of *DB2 Utility Guide and Reference*.
- LOW2KEY and HIGH2KEY columns are limited to storing the first 8 bytes of a key value. If the column is nullable, values are limited to 7 bytes.
- The closer SYSINDEXES.CLUSTERRATIOF is to 100% (a value of 1), the more closely the ordering of the index entries matches the physical ordering of the table rows. Refer to Figure 95 on page 782 to see how an index with a high cluster ratio differs from an index with a low cluster ratio.

## Statistics for partitioned table spaces

For a partitioned table space, DB2 keeps statistics separately by partition and also collectively for the entire table space. If you run RUNSTATS for separate partitions of a table space, DB2 uses the results to update the aggregate statistics for the entire table space.

The list below names the catalog tables that contain statistics by partition and, for each one, the table that contains the corresponding aggregate statistics.

| **Statistics by partition are in:** | **Aggregate statistics are in:** |
|---|---|
| SYSTABSTATS | SYSTABLES |
| SYSINDEXSTATS | SYSINDEXES |
| SYSCOLSTATS | SYSCOLUMNS |
| SYSCOLDISTSTATS | SYSCOLDIST |

***Recommendation:*** Before you run RUNSTATS on separate partitions, run RUNSTATS once on the entire object to generate statistics for all partitions and also aggregate statistics for the entire table space. Alternatively, if you do not plan to load every partition, you can use the FORCEROLLUP YES option or the STATISTICS ROLLUP field on installation panel DSNTIPO to avoid running RUNSTATS on the entire object first. These options force the roll up of the aggregate statistics when statistics do not exist for each separate partition.

## Setting default statistics for created temporary tables

When preparing an SQL statement that refers to a created temporary table, if the table has been instantiated, DB2 uses the cardinality and number of pages maintained for that table in storage. If the table has not been instantiated, DB2 looks at the CARDF and NPAGES columns of the SYSTABLES row for the created temporary table. These values are normally -1 because RUNSTATS cannot run against a created temporary table.

You can establish default statistical values for the cardinality and number of pages if you can estimate the normal cardinality and number of pages that used the values for a particular created temporary table. You can manually update the values in the CARDF and NPAGES columns of the SYSTABLES row for the created temporary table. These values become the default values used if more accurate values are not available or more accurate values cannot be used. The more accurate values are available only for dynamic SQL statements that are prepared after the instantiation of the created temporary table, but within the same unit of work. These more accurate values are not used if the result of the dynamic bind is destined for the Dynamic Statement Cache.

## History statistics

Several catalog tables provide historical statistics for other catalog tables. These catalog history tables include:

- SYSIBM.SYSCOLDIST_HIST
- SYSIBM.SYSCOLUMNS_HIST
- SYSIBM.SYSINDEXES_HIST
- SYSIBM.SYSINDEXPART_HIST
- SYSIBM.SYSINDEXSTATS_HIST
- SYSIBM.SYSLOBSTATS_HIST
- SYSIBM.SYSTABLEPART_HIST
- SYSIBM.SYSTABLES_HIST
- SYSIBM.SYSTABSTATS_HIST

For instance, SYSIBM.SYSTABLESPACE_HIST provides statistics for activity in SYSIBM.SYSTABLESPACE, SYSIBM.SYSTABLEPART_HIST provides statistics for activity in SYSIBM.SYTABLEPART, and so on.

When DB2 adds or changes rows in a catalog table, DB2 might also write information about the rows to the corresponding catalog history table. Although the catalog history tables are not identical to their counterpart tables, they do contain the same columns for access path information and space utilization information. The history statistics provide a way to study trends, to determine when utilities, such as REORG, should be run for maintenance, and to aid in space management.

Table 110 lists the catalog data that are collected for historical statistics. For information on how to gather these statistics, see “Gathering monitor and update statistics” on page 775.

*Table 110. Catalog data collected for historical statistics*

| Column name | Provides access path statistics[1] | Provides space statistics | Description |
|---|---|---|---|
| **SYSIBM.SYSCOLDIST_HIST** | | | |
| CARDF | Yes | No | Number of distinct values gathered |
| COLGROUPCOLNO | Yes | No | Identifies the columns involved in multi-column statistics |
| COLVALUE | Yes | No | Frequently occuring value in the key distribution |
| FREQUENCYF | Yes | No | A number, which multiplied by 100, gives the percentage of rows that contain the value of COLVALUE |
| NUMCOLUMNS | Yes | No | Number of columns involved in multi-column statistics |
| TYPE | Yes | No | Type of statistics gathered, either cardinality (c) or frequent value (F) |
| **SYSIBM.SYCOLUMNS_HIST** | | | |
| COLCARDF | Yes | No | Estimated number of distinct values in the column |

*Table 110. Catalog data collected for historical statistics (continued)*

| Column name | Provides access path statistics[1] | Provides space statistics | Description |
|---|---|---|---|
| HIGH2KEY | Yes | No | Second highest value of the column, or blank |
| LOW2KEY | Yes | No | Second lowest value of the column, or blank |
| **SYSIBM.SYSINDEXES_HIST** | | | |
| CLUSTERING | Yes | No | Whether the index was created with CLUSTER |
| CLUSTERRATIOF | Yes | No | A number, when multiplied by 100, gives the percentage of rows in the clustering order |
| FIRSTKEYCARDF | Yes | No | Number of distinct values in the first key column |
| FULLKEYCARDF | Yes | No | Number of distinct values in the full key |
| NLEAF | Yes | No | Number of active leaf pages |
| NLEVELS | Yes | No | Number of levels in the index tree |
| **SYSIBM.SYSINDEXPART_HIST** | | | |
| CARDF | No | Yes | Number of rows or LOBs referenced |
| DSNUM | No | Yes | Number of data sets |
| EXTENTS | No | Yes | Number of data set extents (when there are multiple pieces, the value is for the extents in the last data set) |
| FAROFFPOSF | No | Yes | Number of rows referenced far from the optimal position |
| LEAFDIST | No | Yes | 100 times the number of pages between successive leaf pages |
| LEAFFAR | No | Yes | Number of leaf pages located physically far away from previous leaf pages for successive active leaf pages accessed in an index scan |
| LEAFNEAR | No | Yes | Number of leaf pages located physically near previous leaf pages for successive active leaf pages |
| NEAROFFPOSF | No | Yes | Number of rows referenced near but not at the optimal position |
| PQTY | No | Yes | Primary space allocation in 4K blocks for the data set |
| PSEUDO_DEL_ENTRIES | No | Yes | Number of pseudo deleted keys |
| SECQTYI | No | Yes | Secondary space allocation in 4K blocks for the data set. |
| SPACEF | No | Yes | Kilobytes of disk storage |
| **SYSIBM.SYSINDEXSTATS_HIST** | | | |
| CLUSTERRATIO | Yes | No | A number, which when multiplied by 100, gives the percentage of rows in the clustering order |
| FIRSTKEYCARDF | Yes | No | Number of distinct values of the first key column |
| FULLKEYCARDF | Yes | No | Number of distinct values of the full key |
| KEYCOUNTF | Yes | No | Total number of rows in the partition |
| NLEAF | Yes | No | Number of leaf pages |
| NLEVELS | Yes | No | Number of levels in the index tree |

*Table 110. Catalog data collected for historical statistics (continued)*

| Column name | Provides access path statistics[1] | Provides space statistics | Description |
|---|---|---|---|
| **SYSIBM.SYSLOBSTATS_HIST** | | | |
| FREESPACE | No | Yes | The amount of free space in the LOB table space |
| ORGRATIO | No | Yes | The ratio of organization in the LOB table space |
| **SYSIBM.SYSTABLEPART_HIST** | | | |
| CARDF | No | Yes | Number of rows in the table space or partition |
| DSNUM | No | Yes | Number of data sets |
| EXTENTS | No | Yes | Number of data set extents (when there are multiple pieces, the value is for the extents in the last data set) |
| FARINDREF | No | Yes | Number of rows relocated far from their original position |
| NEARINDREF | No | Yes | Number of rows relocated near their original position |
| PAGESAVE | No | Yes | Percentage of pages saved by data compression |
| PERCACTIVE | No | Yes | Percentage of space occupied by active pages |
| PERCDROP | No | Yes | Percentage of space occupied by pages from dropped tables |
| PQTY | No | Yes | Primary space allocation in 4K blocks for the data set |
| SECQTYI | No | Yes | Secondary space allocation in 4K blocks for the data set. |
| SPACEF | No | Yes | The number of kilobytes of space currently used |
| **SYSIBM.SYSTABLES_HIST** | | | |
| AVGROWLEN | No | Yes | Average row length of the table specified in the table space |
| CARDF | Yes | No | Number of rows in the table or number of LOBs in an auxiliary table |
| NPAGESF | Yes | No | Number of pages used by the table |
| PCTPAGES | No | Yes | Percentage of pages that contain rows |
| PCTROWCOMP | Yes | No | Percentage of active rows compressed |
| **SYSIBM.SYSTABSTATS_HIST** | | | |
| CARDF | Yes | No | Number of rows in the partition |
| NPAGES | Yes | No | Total number of pages with rows |

[1] The access path statistics in the history tables are collected for historical purposes and are not used for access path selection.

## Gathering monitor and update statistics

The DB2 utility RUNSTATS can update the DB2 catalog tables with statistical information about data and indexes. For a list of the catalog columns for which RUNSTATS collects statistics, see Table 109 on page 766. For instructions on using RUNSTATS, see Part 2 of *DB2 Utility Guide and Reference*.

You can choose which DB2 catalog tables you want RUNSTATS to update: those used to optimize the performance of SQL statements or those used by database administrators to assess the status of a particular table space or index. You can monitor these catalog statistics in conjunction with EXPLAIN to make sure that your queries access data efficiently.

After you use the LOAD, REBUILD INDEX, or REORG utilities, you can gather statistics inline with those utilities by using the STATISTICS option.

***Why gather statistics:*** Maintaining your statistics is a critical part of performance monitoring and tuning. DB2 must have correct statistical information to make the best choices for the access path.

***When to gather statistics:*** To ensure that information in the catalog is current, gather statistics in situations in which the data or index changes significantly, such as in the following situations:

- After loading a table and before binding application plans and packages that access the table.
- After creating an index with the CREATE INDEX statement, to update catalog statistics related to the new index. (Before an application can use a new index, you must rebind the application plan or package.)
- After reorganizing a table space or an index. Then rebind plans or packages for which performance remains a concern. See “Whether to rebind after gathering statistics” on page 786 for more information. (It is not necessary to rebind after reorganizing a LOB table space, because those statistics are not used for access path selection.)
- After heavy insert, update, and delete activity. Again, rebind plans or packages for which performance is critical.
- Periodically. By comparing the output of one execution with previous executions, you can detect a performance problem early.
- Against the DB2 catalog to provide DB2 with more accurate information for access path selection of users’ catalog queries.

To obtain information from the catalog tables, use a SELECT statement, or specify REPORT YES when you invoke RUNSTATS. When used routinely, RUNSTATS provides data about table spaces and indexes over a period of time. For example, when you create or drop tables or indexes or insert many rows, run RUNSTATS to update the catalog. Then rebind your applications so that DB2 can choose the most efficient access paths.

***Collecting statistics by partition:*** You can collect statistics for a single data partition or index partition. This information allows you to avoid the cost of running utilities against unchanged partitions. When you run utilities by partition, DB2 uses the results to update the aggregate statistics for the entire table space or index. If statistics do not exist for each separate partition, DB2 can calculate the aggregate statistics only if the utilities are executed with the FORCEROLLUP YES keyword (or FORCEROLLUP keyword is omitted and the value of the STATISTICS ROLLUP field on installation panel DSNTIPO is YES). If you do not use the keyword or installation panel field setting to force the roll up of the aggregate statistics, you must run utilities once on the entire object before running utilities on separate partitions.

***Collecting history statistics:*** When you collect statistics with RUNSTATS or gather them inline with the LOAD, REBUILD, or REORG utilities, you can use the

HISTORY option to collect history statistics. With the HISTORY option, the utility stores the statistics that were updated in the catalog tables in history records in the corresponding catalog history tables. (For information on the catalog data that is collected for history statistics, seeTable 110 on page 773.)

To remove old statistics that are no longer needed in the catalog history tables, use the MODIFY STATISTICS utility or the SQL DELETE statement. Deleting outdated information from the catalog history tables can help improve the performance of processes that access the data in these tables.

***Recommendations for performance:***

- To reduce the processor consumption WHEN collecting column statistics, use the SAMPLE option. The SAMPLE option allows you to specify a percentage of the rows to examine for column statistics. Consider the effect on access path selection before choosing sampling. There is likely to be little or no effect on access path selection if the access path has a matching index scan and very few predicates. However, if the access path joins of many tables with matching index scans and many predicates, the amount of sampling can affect the access path. In these cases, start with 25 percent sampling and see if there is a negative effect on access path selection. If not, you could consider reducing the sampling percent until you find the percent that gives you the best reduction in processing time without negatively affecting the access path.
- To reduce the elapsed time of gathering statistics immediately after a LOAD, REBUILD INDEX, or REORG, gather statistics inline with those utilities by using the STATISTICS option.

## Updating the catalog

If you have sufficient privileges, you can change all of the values listed in Table 109 on page 766 by executing SQL UPDATE statements.

***Running RUNSTATS after UPDATE:*** If you change values in the catalog and later run RUNSTATS to update those values, your changes are lost.

***Recommendation:*** Keep track of the changes you make and of the plans or packages that have an access path change due to changed statistics.

## Correlations in the catalog

The following relationships exist among certain columns of the catalog tables:

- Columns within table SYSCOLUMNS
- Columns in the tables SYSCOLUMNS and SYSINDEXES
- Columns in the tables SYSCOLUMNS and SYSCOLDIST
- Columns in the tables SYSCOLUMNS, SYSCOLDIST, and SYSINDEXES
- Columns with table space statistics and columns for partition-level statistics, as described in “Statistics for partitioned table spaces” on page 772.

If you plan to update some values, keep in mind the following correlations:

- COLCARDF and FIRSTKEYCARDF. For a column that is the first column of an index, those two values are equal. If the index has only that one column, the two values are also equal to the value of FULLKEYCARDF.
- COLCARDF, LOW2KEY, and HIGH2KEY. If the COLCARDF value is not ’-1’, DB2 assumes that statistics exist for the column. In particular, it uses the values of LOW2KEY and HIGH2KEY in calculating filter factors.

- The CARDF column in SYSCOLDIST is related to COLCARDF in SYSIBM.SYSCOLUMNS and to FIRSTKEYCARDF and FULLKEYCARDF in SYSIBM.SYSINDEXES. CARDF must be the minimum of the following:
  - A value between FIRSTKEYCARDF and FULLKEYCARDF if the index contains the same set of columns
  - A value between MAX(COLCARDF of each column in the column group) and the product of multiplying together the COLCARDF of each column in the column group

  For example, assume a set of statistics as shown in Figure 94. The range between FIRSTKEYCARDF and FULLKEYCARDF of 100 and 10 000. The maximum of the COLCARDF values is 50 000. Thus, the allowable range is between 100 and 10 000.

```
CARDF = 1000
NUMCOLUMNS = 3
COLGROUPCOLNO = 2,3,5

INDEX1 on columns 2,3,5,7,8
FIRSTKEYCARDF = 100                CARDF must be between 100
FULLKEYCARDF = 10000                                and 10000

column 2 COLCARDF = 100
column 3 COLCARDF =  50
column 5 COLCARDF =  10
```

*Figure 94. Determining valid values for CARDF. In this example, CARDF is bounded by 100 and 10 000.*

## Recommendation for COLCARDF and FIRSTKEYCARDF

On partitioned indexes, RUNSTATS INDEX calculates the number of distinct column values and saves it in SYSCOLSTATS.COLCARD by partition. When the statistics by partition are used to form the aggregate, the aggregate might not be exact because some column values could occur in more than one partition. Without scanning all parts of the index, DB2 cannot detect that overlap. The overlap never skews COLCARD by more than the number of partitions, which should not be a problem for large values. For small values, you might want to update the aggregate COLCARDF value in SYSCOLUMNS, because DB2 uses the COLCARD value when determining access paths.

The exception and remedy described above for COLCARD and COLCARDF is also true for the FIRSTKEYCARDF column in SYSIBM.SYSINDEXES and the FIRSTKEYCARDF column in SYSIBM.SYSINDEXSTATS.

## Recommendation for HIGH2KEY and LOW2KEY

If you update the COLCARDF value for a column, also update HIGH2KEY and LOW2KEY for the column. HIGH2KEY and LOW2KEY are defined as CHAR(8); thus, an UPDATE statement must provide a character or hexadecimal value. Entering a character value is quite straightforward: SET LOW2KEY = 'ALAS', for instance. But to enter a numeric, date, or time value you must use the hexadecimal value of the DB2 internal format. See “Internal formats for dates, times, and timestamps” on page 954 and “DB2 codes for numeric data” on page 955. Be sure to allow for a null indicator in keys that allow nulls. See also “Null values” on page 952.

## Statistics for distributions

Statistics for distributions are stored in the catalog tables SYSCOLDIST and SYSCOLDISTSTATS. By default, DB2 inserts the 10 most frequent values as well as the first and last key values. See Part 2 of *DB2 Utility Guide and Reference* for information about collecting more statistics related to columns that are correlated.

You can insert, update, or delete distribution information for any column, whether or not it is a first key column of an index. But to enter a numeric, date, or time value you must use the hexadecimal value of the DB2 internal format. See “Internal formats for dates, times, and timestamps” on page 954 and “DB2 codes for numeric data” on page 955. Be sure to allow for a null indicator in keys that allow nulls. See also “Null values” on page 952.

## Recommendation for using the TIMESTAMP column

Statistics gathered by RUNSTATS include timestamps. Every row updated or inserted during a particular invocation of RUNSTATS contains the same timestamp value. Update column STATSTIME whenever you update statistics in the catalog, so that you can always determine when they were last updated.

# Querying the catalog for statistics

The SELECT statements below show you how to retrieve some of the important statistics for access path selection. The catalog queries shown here are included in DSNTESP in SDSNSAMP and can be used as input to SPUFI. See “Chapter 33. Using EXPLAIN to improve SQL performance” on page 789 for more information about how these statistics are used in access path selection. See Appendix D of *DB2 SQL Reference* for the table definitions and descriptions of all DB2 catalog tables.

**Product-sensitive Programming Interface**

To access information about your data and how it is organized, use the following queries:

```
SELECT CREATOR, NAME, CARDF, NPAGES, PCTPAGES
  FROM SYSIBM.SYSTABLES
  WHERE DBNAME = 'xxx'
  AND TYPE = 'T';
```

```
SELECT NAME, UNIQUERULE, CLUSTERRATIOF, FIRSTKEYCARDF, FULLKEYCARDF,
  NLEAF, NLEVELS, PGSIZE
  FROM SYSIBM.SYSINDEXES
  WHERE DBNAME = 'xxx';
```

```
SELECT NAME, DBNAME, NACTIVE, CLOSERULE, LOCKRULE
  FROM SYSIBM.SYSTABLESPACE
  WHERE DBNAME = 'xxx';
```

```
SELECT NAME, TBNAME, COLCARDF, HIGH2KEY, LOW2KEY, HEX(HIGH2KEY),
  HEX(LOW2KEY)
  FROM SYSIBM.SYSCOLUMNS
  WHERE TBCREATOR = 'xxx' AND COLCARDF <> -1;
```

```
SELECT NAME, FREQUENCYF, COLVALUE, HEX(COLVALUE), CARDF,
  COLGROUPCOLNO, HEX(COLGROUPCOLNO), NUMCOLUMNS, TYPE
  FROM SYSIBM.SYSCOLDIST
  WHERE TBNAME = 'ttt'
  ORDER BY NUMCOLUMNS, NAME, COLGROUPCOLNO, TYPE, FREQUENCYF DESC;
```

```
SELECT NAME, TSNAME, CARD, NPAGES
   FROM SYSIBM.SYSTABSTATS
   WHERE DBNAME='xxx';
```

**End of Product-sensitive Programming Interface**

If the statistics in the DB2 catalog no longer correspond to the true organization of your data, you should reorganize the necessary tables, run RUNSTATS, and rebind the plans or packages that contain any affected queries. See “When to reorganize indexes and table spaces” on page 784 and the description of REORG in Part 2 of *DB2 Utility Guide and Reference* for information on how to determine which table spaces and indexes qualify for reorganization. This includes the DB2 catalog table spaces as well as user table spaces. Then DB2 has accurate information to choose appropriate access paths for your queries. Use the EXPLAIN statement to verify the chosen access paths for your queries.

## Improving index and table space access

Statistics from the DB2 catalog help determine the most economical access path. The statistics described in this section are used to determine index access cost and are found in the corresponding columns of the SYSIBM.SYSINDEXES catalog table.

The statistics show distribution of data within the allocated space, from which you can judge clustering and the need to reorganize.

Space utilization statistics can also help you make sure that access paths that use the index or table space are as efficient as possible. By reducing gaps between leaf pages in an index, or to ensure that data pages are close together, you can reduce sequential I/Os.

To provide the most accurate data, gather statistics routinely to provide data about table spaces and indexes over a period of time. One recommendation is to run

RUNSTATS some time *after* reorganizing the data or indexes. By gathering the statistics after you reorganize, you ensure that access paths reflect a more “average” state of the data.

This section describes the following topics:

- “How clustering affects access path selection”
- “What other statistics provide index costs” on page 783
- “When to reorganize indexes and table spaces” on page 784
- “Whether to rebind after gathering statistics” on page 786

## How clustering affects access path selection

In general, CLUSTERRATIOF gives an indication of how closely the order of the index entries on the index leaf pages matches the actual ordering of the rows on the data pages. The closer CLUSTERRATIOF is to 100%, the more closely the ordering of the index entries matches the actual ordering of the rows on the data pages. The actual formula is quite complex and accounts for indexes with many duplicates; in general, for a given index, the more duplicates, the higher the CLUSTERRATIOF value.

Here are some things to remember about the effect of CLUSTERRATIOF on access paths:

- CLUSTERRATIOF is an important input to the cost estimates that are used to determine whether an index is used for an access path, and, if so, which index to use.
- If the access is INDEXONLY, then this value does not apply.
- The higher the CLUSTERRATIOF value, the lower the cost of referencing data pages during an index scan is.
- For an index that has a CLUSTERRATIOF less than 80%, sequential prefetch is not used to access the data pages.

Figure 95 on page 782 shows an index scan on an index with a high cluster ratio. Compare that with Figure 96 on page 783, which shows an index scan on an index with a low cluster ratio.

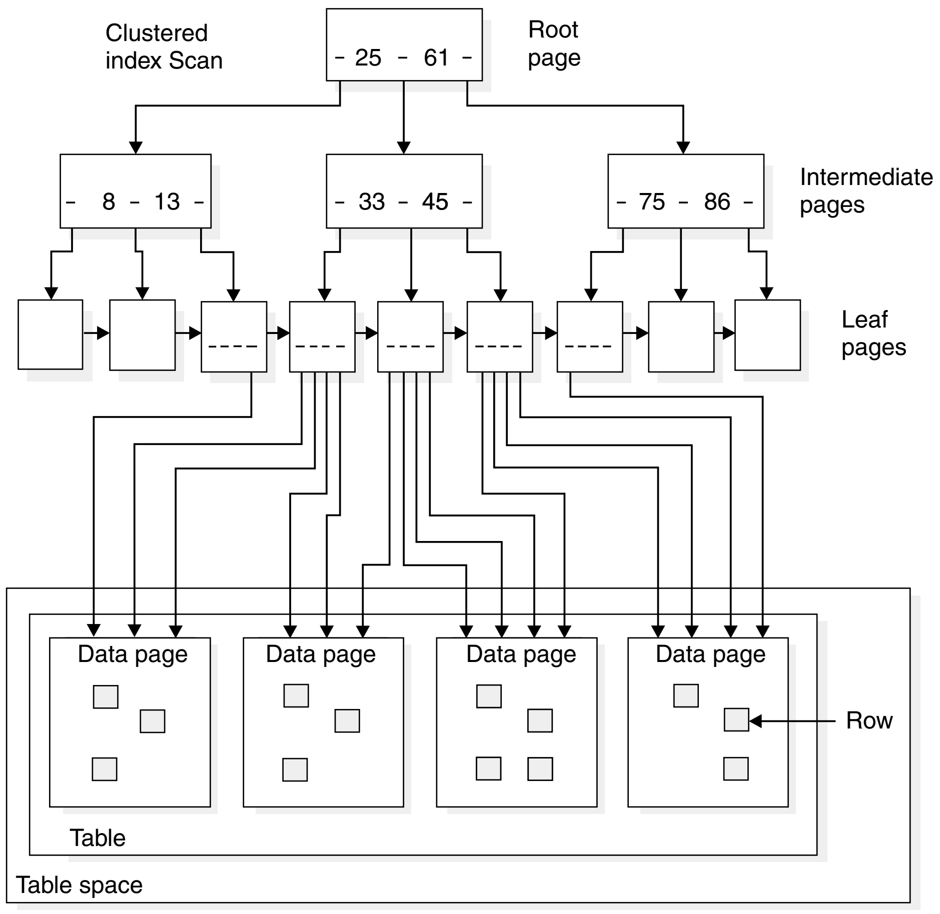


*Figure 95. A clustered index scan. This figure assumes that the index is 100% clustered*

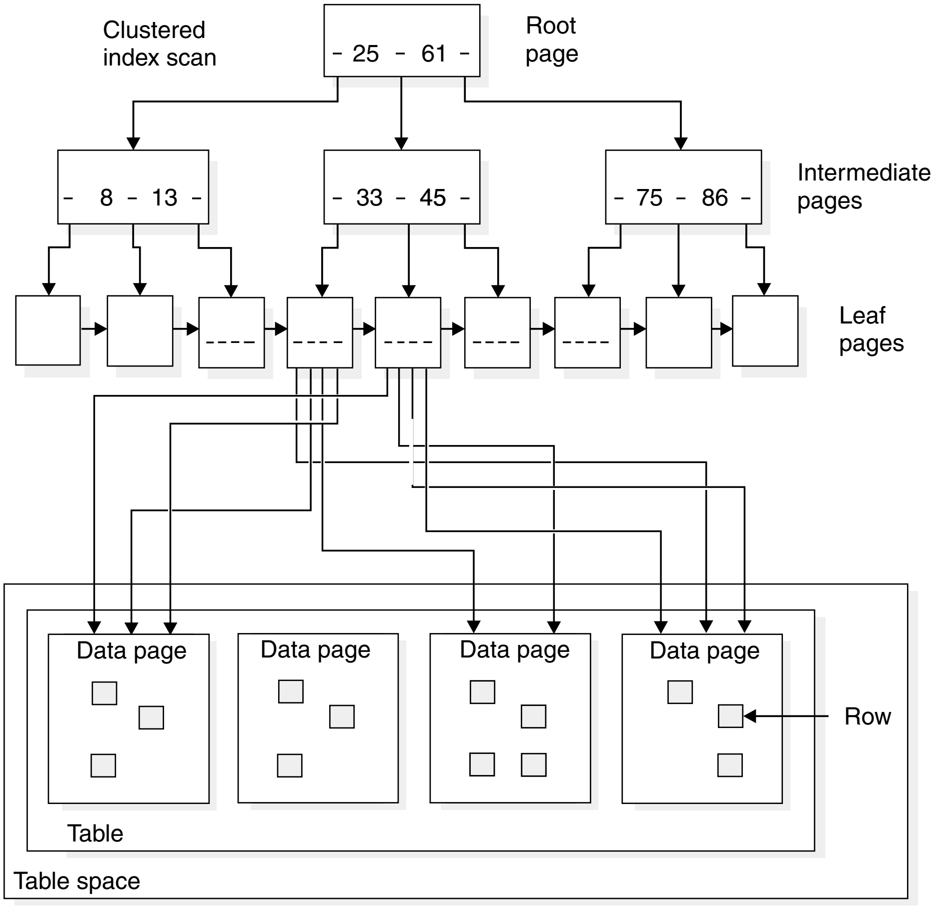


*Figure 96. A nonclustered index scan. In some cases, DB2 can access the data pages in order even when a nonclustered index is used.*

## What other statistics provide index costs

The following statistics in SYSINDEXES also give information about costs to process the index.

***FIRSTKEYCARDF:*** The number of distinct values of the first index key column. When an indexable equal predicate is specified on the first index key column, 1/FIRSTKEYCARDF is the filter factor for the predicate and the index. The higher the number is, the less the cost is.

***FULLKEYCARDF:*** The number of distinct values for the entire index key. When indexable equal predicates are specified on all the index key columns, 1/FULLKEYCARDF is the filter factor for the predicates and the index. The higher the number is, the less the cost is.

When the number of matching columns is greater than 1 and less than the number of index key columns, the filtering of the index is located between 1/FIRSTKEYCARDF and 1/FULLKEYCARDF.

***NLEAF:*** The number of active leaf pages in the index. NLEAF is a portion of the cost to scan the index. The smaller the number is, the less the cost is. It is also less when the filtering of the index is high, which comes from FIRSTKEYCARDF, FULLKEYCARDF, and other indexable predicates.

***NLEVELS:*** The number of levels in the index tree. NLEVELS is another portion of the cost to traverse the index. The same conditions as NLEAF apply. The smaller the number is, the less the cost is.

## When to reorganize indexes and table spaces

Data that is organized well physically can improve the performance of access paths that rely on index or data scans. Well-organized data can also help reduce the amount of disk storage used by the index or table space. If your main reason for reorganizing is performance, the best way to determine when to reorganize is to watch your statistics for increased I/O, getpages, and processor consumption. When performance degrades to an unacceptable level, analyze the statistics described in the rules of thumb in this section to help you develop your own rules for when to reorganize in your particular environment. Here are some general rules of thumb for when to consider running REORG. See Part 2 of *DB2 Utility Guide and Reference* for more information.

***Using useful catalog queries:*** Catalog queries you can use to help you determine when to reorganize are included in DSNTESP in SDSNSAMP and can be used as input to SPUFI.

***Using REORG to determine whether to reorganize:*** The REORG utility imbeds the function of catalog queries. If a query returns a certain result (you can use the default or supply your own), REORG will either reorganize or not. Optionally, you can have REORG run a report instead of actually doing the reorganization.

The REORG options that imbed these catalog queries are:

- OFFPOSLIMIT and INDREFLIMIT options of REORG TABLESPACE
- LEAFDISTLIMIT of REORG INDEX

REORG does not imbed any function to help you determine when to reorganize LOB table spaces.

### Reorganizing Indexes

To understand index organization, you must understand the LEAFNEAR and LEAFFAR columns of SYSIBM.SYSINDEXPART. This section describes how to interpret those values and then describes some rules of thumb for determining when to reorganize the index.

***Understanding LEAFNEAR and LEAFFAR:*** The LEAFNEAR and LEAFFAR columns of SYSIBM.SYSINDEXPART measure the disorganization of physical leaf pages by indicating the number of pages that are not in an optimal position. Leaf pages can have page gaps whenever index pages are deleted or when there are index leaf page splits caused by an insert that cannot fit onto a full page. If the key cannot fit on the page, DB2 moves half the index entries onto a new page, which might be far away from the “home” page.

Figure 97 on page 785 shows the logical and physical view of an index.

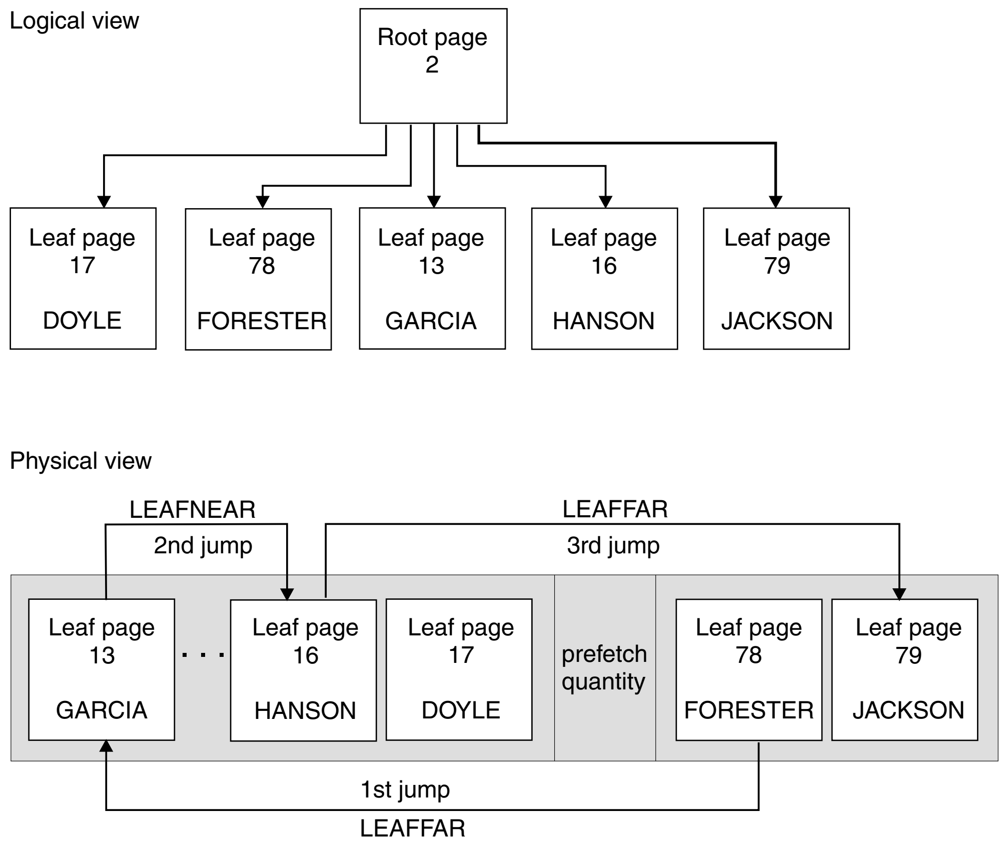


*Figure 97. Logical and physical views of an index in which LEAFNEAR=1 and LEAFFAR=2*

The logical view at the top of the figure shows that for an index scan four leaf pages need to be scanned to access the data for FORESTER through JACKSON. The physical view at the bottom of the figure shows how the pages are physically accessed. The first page is at physical leaf page 78, and the other leaf pages are at physical locations 79, 13, and 16. A jump forward or backward of more than one page represents non-optimal physical ordering. LEAFNEAR represents the number of jumps within the prefetch quantity, and LEAFFAR represents the number of jumps outside the prefetch quantity. In this example, assuming that the prefetch quantity is 32, there are two jumps outside the prefetch quantity—a jump from page 78 to page 13, and one from page 16 to page 79. Thus, LEAFFAR is 2. Because of the jump within the prefetch quantity from page 13 to page 16, LEAFNEAR is 1.

LEAFNEAR has a smaller impact than LEAFFAR because the LEAFNEAR pages, which are located within the prefetch quantity, are typically read by prefetch without incurring extra I/Os.

The optimal value of the LEAFNEAR and LEAFFAR catalog columns is zero. However, immediately after you run the REORG and gather statistics, LEAFNEAR for a large index might be greater than zero. A non-zero value could be caused by free pages that result from the FREEPAGE option on CREATE INDEX, non-leaf pages, or various system pages; the jumps over these pages are included in LEAFNEAR.

***Rules of thumb:*** Consider running REORG INDEX in the following cases:

- LEAFFAR ⁄ NLEAF is greater than 10%.

  NLEAF is a column in SYSIBM.SYSINDEXES.

- PSEUDO_DEL_ENTRIES ⁄ CARDF is greater than 10%.

  If you are reorganizing the index because of this value, consider using the REUSE option to improve performance.
- When the data set has multiple extents. 50 extents is a general guideline.

  Many secondary extents can detract from performance of index scans because the data on those extents is not necessarily physically located near the rest of the index data.

### Reorganizing table spaces

SYSIBM.SYSTABLEPART contains information about how the data in the table space is physically stored. Consider running REORG TABLESPACE in the following situations:

- FAROFFPOSF ⁄ CARDF is greater than 10%. Or, if the index is a clustering index, the CLUSTERRATIOF column of SYSIBM.SYSINDEXES is less than 90%.
- (NEARINDREF + FARINDREF) ⁄ CARDF is greater than 10%.
- PERCDROP is greater than 10% for a simple table space.

  If you are reorganizing the table space because of this value, consider using the REUSE option to improve performance.
- The data set has multiple extents. 50 extents is a general guideline.

### Reorganizing LOB table spaces

SYSIBM.SYSLOBSTATS contains information about how the data in the table space is physically stored. Consider running REORG on the LOB table space when the value in the ORGRATIO column is 2.

## Whether to rebind after gathering statistics

It is not always necessary to rebind all applications after you gather statistics. A rebind is necessary only if the access path statistics change significantly from the last time you bound the applications and if performance suffers as a result.

When performance degrades to an unacceptable level, analyze the statistics described in the rules of thumb in this section to help you develop your own guidelines for when to rebind.

Consider the following rules of thumb about when to rebind:

- CLUSTERRATIOF changes to less or more than 80% (a value of 0.80).
- NLEAF changes more than 20% from the previous value.
- NLEVELS changes (only if it was more than a 2-level index to begin with).
- NPAGES changes more than 20% from the previous value.
- NACTIVEF changes more than 20% from the previous value.
- The range of HIGH2KEY to LOW2KEY range changes more than 20% from the range previously recorded.
- Cardinality changes more than 20% from previous range.
- Distribution statistics change the majority of the frequent column values.

## Modeling your production system

To see what access paths your production queries will use, consider updating the catalog statistics on your test system to be the same as your production system.

To do that, run RUNSTATS on your production tables to get current statistics for access path selection. Then retrieve them and use them to build SQL statements to update the catalog of the test system. You can use queries similar to those in Figure 98 to build those statements.

**Product-sensitive Programming Interface**

```
SELECT DISTINCT 'UPDATE SYSIBM.SYSTABLESPACE SET NACTIVEF='
CONCAT DIGITS(DECIMAL(NACTIVEF,31,0))
CONCAT ' WHERE NAME=''' CONCAT TS.NAME
CONCAT ''' AND CREATOR =''' CONCAT TS.CREATOR CONCAT'''*'
FROM SYSIBM.SYSTABLESPACE TS, SYSIBM.SYSTABLES TBL
WHERE TS.NAME = TSNAME
      AND TBL.NAME IN ('table list')
      AND TBL.CREATOR IN ('creator list')
      AND NACTIVE >=0;

SELECT 'UPDATE SYSIBM.SYSTABLES SET CARDF='
CONCAT  DIGITS(DECIMAL(CARDF,31,0))
CONCAT',NPAGES='CONCAT DIGITS(NPAGES)
CONCAT ' WHERE NAME='''CONCAT NAME
CONCAT ''' AND CREATOR ='''CONCAT CREATOR CONCAT'''*'
FROM SYSIBM.SYSTABLES
WHERE NAME IN ('table list')
      AND CREATOR IN ('creator list')
      AND CARDF >= 0;

SELECT 'UPDATE SYSIBM.SYSINDEXES SET FIRSTKEYCARDF='
CONCAT DIGITS(DECIMAL(FIRSTKEYCARDF,31,0))
CONCAT ',FULLKEYCARDF='CONCAT DIGITS(DECIMAL(FULLKEYCARDF,31,0))
CONCAT',NLEAF='CONCAT DIGITS(NLEAF)
CONCAT',NLEVELS='CONCAT DIGITS(NLEVELS)
CONCAT',CLUSTERRATIOF='CONCAT DIGITS(DECIMAL(CLUSTERRATIOF,31,0))
CONCAT' WHERE NAME='''CONCAT NAME
CONCAT ''' AND CREATOR ='''CONCAT CREATOR CONCAT'''*'
FROM SYSIBM.SYSINDEXES
WHERE TBNAME IN ('table list')
      AND CREATOR IN ('creator list')
          AND FULLKEYCARDF >= 0;

SELECT 'UPDATE SYSIBM.SYSCOLUMNS SET COLCARDF='
CONCAT DIGITS(DECIMAL(COLCARDF,31,0))
CONCAT',HIGH2KEY=''' CONCAT HIGH2KEY
CONCAT''',LOW2KEY=''' CONCAT LOW2KEY
CONCAT''' WHERE TBNAME=''' CONCAT TBNAME CONCAT  ''' AND COLNO='
CONCAT DIGITS(COLNO)
CONCAT ' AND TBCREATOR =''' CONCAT TBCREATOR CONCAT'''*'
FROM SYSIBM.SYSCOLUMNS
WHERE TBNAME IN ('table list')
      AND TBCREATOR IN ('creator list')
      AND COLCARDF >= 0;
```

**End of Product-sensitive Programming Interface**

*Figure 98. Statements to generate update statistics on test system (Part 1 of 2)*

**Product-sensitive Programming Interface**

```
DELETE * FROM (test_system).SYSCOLDIST;

SELECT * FROM (production_system).SYSCOLDIST;

Using values from the production system's SYSCOLDIST table:

INSERT INTO (test_system).SYSCOLDIST;
```

**End of Product-sensitive Programming Interface**

*Figure 98. Statements to generate update statistics on test system (Part 2 of 2)*

***Notes to Figure 98 on page 787:***

- The third SELECT is 215 columns wide; you might need to change your default character column width if you are using SPUFI.
- Asterisks (*) appear in the examples to avoid having the semicolon interpreted as the end of the SQL statement. Edit the result to change the asterisk to a semicolon.

***Access path differences from test to production:*** When you bind applications on the test system with production statistics, access paths should be similar to what you see when the same query is bound on your production system. The access paths from test to production could be different for the following possible reasons:

- The processor models are different.
- The buffer pool sizes are different.
- Data in SYSIBM.SYSCOLDIST is mismatched. (This mismatch occurs only if some of the steps mentioned above were not followed exactly).

***Tools to help:*** If your production system is accessible from your test system, you can use DB2 PM EXPLAIN on your test system to request EXPLAIN information from your production system. This request can reduce the need to simulate a production system by updating the catalog.

You can also use the DB2 Visual Explain feature to display the current PLAN_TABLE output or the graphed access paths for statements within any particular subsystem from your workstation environment. For example, if you have your test system on one subsystem and your production system on another subsystem, you can visually compare the PLAN_TABLE outputs or access paths simultaneously with some window or view manipulation. You can then access the catalog statistics for certain referenced objects of an access path from either of the displayed PLAN_TABLEs or access path graphs. For information on using Visual Explain, see DB2 Visual Explain online help.

# Chapter 33. Using EXPLAIN to improve SQL performance

The information under this heading, up to the end of this chapter, is Product-sensitive Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

***Definitions and purpose:*** EXPLAIN is a monitoring tool that produces information about the following:

- A plan, package, or SQL statement when it is bound. The output appears in a table you create called PLAN_TABLE, which is also called a *plan table*. For experienced users, you can use PLAN_TABLE to give optimization hints to DB2. See “Giving optimization hints to DB2” on page 757 for more information.
- An estimated cost of executing an SQL SELECT, INSERT, UPDATE, or DELETE statement. The output appears in a table you create called DSN_STATEMNT_TABLE, which is also called a *statement table*. For more information about statement tables, see “Estimating a statement’s cost” on page 836.
- User-defined functions referred to in the statement, including the specific name and schema. The output appears in a table you create called DSN_FUNCTION_TABLE, which is also called a *function table*. For more information about function tables, see Part 3 of *DB2 Application Programming and SQL Guide*.

***Other tools:*** The following tools can help you tune SQL queries:

- DB2 Visual Explain

  Visual Explain is a graphical workstation feature of DB2 that provides:

  – An easy-to-understand display of a selected access path
  – Suggestions for changing an SQL statement
  – An ability to invoke EXPLAIN for dynamic SQL statements
  – An ability to provide DB2 catalog statistics for referenced objects of an access path
  – A subsystem parameter browser with keyword 'Find' capabilities

  For information on using DB2 Visual Explain, which is a separately packaged CD-ROM provided with your DB2 Version 7 license, see *DB2 Visual Explain online help*.
- DB2 Performance Monitor (PM)

  DB2 PM is a performance monitoring tool that formats performance data. DB2 PM combines information from EXPLAIN and from the DB2 catalog. It displays access paths, indexes, tables, table spaces, plans, packages, DBRMs, host variable definitions, ordering, table access and join sequences, and lock types. Output is presented in a dialog rather than as a table, making the information easy to read and understand.
- DB2 Estimator

  DB2 Estimator for Windows is an easy-to-use, stand-alone tool for estimating the performance of DB2 for OS/390 and z/OS applications. You can use it to predict the performance and cost of running the applications, transactions, SQL statements, triggers, and utilities. For instance, you can use DB2 Estimator for estimating the impact of adding or dropping an index from a table, estimating the change in response time from adding processor resources, and estimating the amount of time a utility job will take to run. DB2 Estimator for Windows can be downloaded from the Web.

***Chapter overview:*** This chapter includes the following topics:

- “Obtaining PLAN_TABLE information from EXPLAIN”
- “Estimating a statement’s cost” on page 836
- “Asking questions about data access” on page 798
- “Interpreting access to a single table” on page 805
- “Interpreting access to two or more tables (join)” on page 812
- “Interpreting data prefetch” on page 824
- “Determining sort activity” on page 828
- “Processing for views and nested table expressions” on page 829

See also “Chapter 34. Parallel operations and query performance” on page 841.

## Obtaining PLAN_TABLE information from EXPLAIN

The information in PLAN_TABLE can help you to:

- Design databases, indexes, and application programs
- Determine when to rebind an application
- Determine the access path chosen for a query

For each access to a single table, EXPLAIN tells you if an index access or table space scan is used. If indexes are used, EXPLAIN tells you how many indexes and index columns are used and what I/O methods are used to read the pages. For joins of tables, EXPLAIN tells you which join method and type are used, the order in which DB2 joins the tables, and when and why it sorts any rows.

The primary use of EXPLAIN is to observe the access paths for the SELECT parts of your statements. For UPDATE and DELETE WHERE CURRENT OF, and for INSERT, you receive somewhat less information in your plan table. And some accesses EXPLAIN does not describe: for example, the access to LOB values, which are stored separately from the base table, and access to parent or dependent tables needed to enforce referential constraints.

The access paths shown for the example queries in this chapter are intended only to illustrate those examples. If you execute the queries in this chapter on your system, the access paths chosen can be different.

***Steps to obtain PLAN_TABLE information:*** Use the following overall steps to obtain information from EXPLAIN:

1. Have appropriate access to a plan table. To create the table, see “Creating PLAN_TABLE”.
2. Populate the table with the information you want. For instructions, see “Populating and maintaining a plan table” on page 796.
3. Select the information you want from the table. For instructions, see “Reordering rows from a plan table” on page 797.

### Creating PLAN_TABLE

Before you can use EXPLAIN, you must create a table called PLAN_TABLE to hold the results of EXPLAIN. A copy of the statements needed to create the table are in the DB2 sample library, under the member name DSNTESC. (Unless you need the information they provide, it is not necessary to create a function table or statement table to use EXPLAIN.)

Figure 99 shows the format of a plan table. Table 111 on page 792 shows the content of each column.

Your plan table can use many formats, but use the 51-column format because it gives you the most information. If you alter an existing plan table to add new columns, specify the columns as NOT NULL WITH DEFAULT, so that default values are included for the rows already in the table. However, as you can see in Figure 99, certain columns do allow nulls. Do not specify those columns as NOT NULL WITH DEFAULT.

```
      QUERYNO          INTEGER       NOT NULL          PREFETCH          CHAR(1)      NOT NULL WITH DEFAULT
      QBLOCKNO         SMALLINT      NOT NULL          COLUMN_FN_EVAL    CHAR(1)      NOT NULL WITH DEFAULT
      APPLNAME         CHAR(8)       NOT NULL          MIXOPSEQ          SMALLINT     NOT NULL WITH DEFAULT
      PROGNAME         CHAR(8)       NOT NULL            ---------28 column format ---------
      PLANNO           SMALLINT      NOT NULL          VERSION           VARCHAR(64)  NOT NULL WITH DEFAULT
      METHOD           SMALLINT      NOT NULL          COLLID            CHAR(18)     NOT NULL WITH DEFAULT
      CREATOR          CHAR(8)       NOT NULL            ---------30 column format ---------
      TNAME            CHAR(18)      NOT NULL          ACCESS_DEGREE     SMALLINT
      TABNO            SMALLINT      NOT NULL          ACCESS_PGROUP_ID  SMALLINT
      ACCESSTYPE       CHAR(2)       NOT NULL          JOIN_DEGREE       SMALLINT
      MATCHCOLS        SMALLINT      NOT NULL          JOIN_PGROUP_ID    SMALLINT
      ACCESSCREATOR    CHAR(8)       NOT NULL            ---------34 column format ---------
      ACCESSNAME       CHAR(18)      NOT NULL          SORTC_PGROUP_ID   SMALLINT
      INDEXONLY        CHAR(1)       NOT NULL          SORTN_PGROUP_ID   SMALLINT
      SORTN_UNIQ       CHAR(1)       NOT NULL          PARALLELISM_MODE  CHAR(1)
      SORTN_JOIN       CHAR(1)       NOT NULL          MERGE_JOIN_COLS   SMALLINT
      SORTN_ORDERBY    CHAR(1)       NOT NULL          CORRELATION_NAME  CHAR(18)
      SORTN_GROUPBY    CHAR(1)       NOT NULL          PAGE_RANGE        CHAR(1)      NOT NULL WITH DEFAULT
      SORTC_UNIQ       CHAR(1)       NOT NULL          JOIN_TYPE         CHAR(1)      NOT NULL WITH DEFAULT
      SORTC_JOIN       CHAR(1)       NOT NULL          GROUP_MEMBER      CHAR(8)      NOT NULL WITH DEFAULT
      SORTC_ORDERBY    CHAR(1)       NOT NULL          IBM_SERVICE_DATA  VARCHAR(254) NOT NULL WITH DEFAULT
      SORTC_GROUPBY    CHAR(1)       NOT NULL            --------43 column format ----------
      TSLOCKMODE       CHAR(3)       NOT NULL          WHEN_OPTIMIZE     CHAR(1)      NOT NULL WITH DEFAULT
      TIMESTAMP        CHAR(16)      NOT NULL          QBLOCK_TYPE       CHAR(6)      NOT NULL WITH DEFAULT
      REMARKS          VARCHAR(254)  NOT NULL          BIND_TIME         TIMESTAMP    NOT NULL WITH DEFAULT
        ---------25 column format ---------              ------46 column format ------------
                                                       OPTHINT            CHAR(8)     NOT NULL WITH DEFAULT
                                                       HINT_USED          CHAR(8)     NOT NULL WITH DEFAULT
                                                       PRIMARY_ACCESSTYPE CHAR(1)     NOT NULL WITH DEFAULT
                                                         -------49 column format------------
#                                                      PARENT_QBLOCKNO    SMALLINT    NOT NULL WITH DEFAULT
|                                                      TABLE_TYPE         CHAR(1)
                                                         -------51 column format-----------
```

*Figure 99. Format of PLAN_TABLE*

*Table 111. Descriptions of columns in PLAN_TABLE*

| Column Name | Description |
|---|---|
| QUERYNO | A number intended to identify the statement being explained. For a row produced by an EXPLAIN statement, specify the number in the QUERYNO clause. For a row produced by non-EXPLAIN statements, specify the number using the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE and DELETE statement syntax. Otherwise, DB2 assigns a number based on the line number of the SQL statement in the source program.<br>When the values of QUERYNO are based on the statement number in the source program, values greater than 32767 are reported as 0. Hence, in a very long program, the value is not guaranteed to be unique. If QUERYNO is not unique, the value of TIMESTAMP is unique. |
| QBLOCKNO | The position of the query in the statement being explained (1 for the outermost query, 2 for the next query, and so forth). For better performance, DB2 might merge a query block into another query block. When that happens, the position number of the merged query block will not be in QBLOCKNO. |
| APPLNAME | The name of the application plan for the row. Applies only to embedded EXPLAIN statements executed from a plan or to statements explained when binding a plan. Blank if not applicable. |
| PROGNAME | The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. Blank if not applicable. |
| PLANNO | The number of the step in which the query indicated in QBLOCKNO was processed. This column indicates the order in which the steps were executed. |
| METHOD | A number (0, 1, 2, 3, or 4) that indicates the join method used for the step:<br>**0** First table accessed, continuation of previous table accessed, or not used.<br>**1** *Nested loop* join. For each row of the present composite table, matching rows of a new table are found and joined.<br>**2** *Merge scan* join. The present composite table and the new table are scanned in the order of the join columns, and matching rows are joined.<br>**3** Sorts needed by ORDER BY, GROUP BY, SELECT DISTINCT, UNION, a quantified predicate, or an IN predicate. This step does not access a new table.<br>**4** *Hybrid* join. The current composite table is scanned in the order of the join-column rows of the new table. The new table is accessed using list prefetch. |
| CREATOR | The creator of the new table accessed in this step, blank if METHOD is 3. |
| TNAME | The name of a table, created or declared temporary table, materialized view, or materialized table expresssion. The value is blank if METHOD is 3. The column can also contain the name of a table in the form DSNWFQB(*qblockno*). DSNWFQB(*qblockno*) is used to represent the intermediate result of a UNION ALL or an outer join that is materialized. If a view is merged, the name of the view does not appear.<br>A value of Q in TABLE_TYPE for the name of a view or nested table expresssion indicates that the materialization was virtual and not actual. Materialization can be virtual when the view or nested table expression definition contains a UNION ALL that is not distributed. |
| TABNO | Values are for IBM use only. |

*Table 111. Descriptions of columns in PLAN_TABLE (continued)*

| Column Name | Description |
|---|---|
| ACCESSTYPE | The method of accessing the new table:<br>**I** By an index (identified in ACCESSCREATOR and ACCESSNAME)<br>**I1** By a one-fetch index scan<br>**N** By an index scan when the matching predicate contains the IN keyword<br>**R** By a table space scan<br>**M** By a multiple index scan (followed by MX, MI, or MU)<br>**MX** By an index scan on the index named in ACCESSNAME<br>**MI** By an intersection of multiple indexes<br>**MU** By a union of multiple indexes<br>**blank** Not applicable to the current row |
| MATCHCOLS | For ACCESSTYPE I, I1, N, or MX, the number of index keys used in an index scan; otherwise, 0. |
| ACCESSCREATOR | For ACCESSTYPE I, I1, N, or MX, the creator of the index; otherwise, blank. |
| ACCESSNAME | For ACCESSTYPE I, I1, N, or MX, the name of the index; otherwise, blank. |
| INDEXONLY | Whether access to an index alone is enough to carry out the step, or whether data too must be accessed. Y=Yes; N=No. For exceptions, see “Is the query satisfied using only the index? (INDEXONLY=Y)” on page 800. |
| SORTN_UNIQ | Whether the new table is sorted to remove duplicate rows. Y=Yes; N=No. |
| SORTN_JOIN | Whether the new table is sorted for join method 2 or 4. Y=Yes; N=No. |
| SORTN_ORDERBY | Whether the new table is sorted for ORDER BY. Y=Yes; N=No. |
| SORTN_GROUPBY | Whether the new table is sorted for GROUP BY. Y=Yes; N=No. |
| SORTC_UNIQ | Whether the composite table is sorted to remove duplicate rows. Y=Yes; N=No. |
| SORTC_JOIN | Whether the composite table is sorted for join method 1, 2 or 4. Y=Yes; N=No. |
| SORTC_ORDERBY | Whether the composite table is sorted for an ORDER BY clause or a quantified predicate. Y=Yes; N=No. |
| SORTC_GROUPBY | Whether the composite table is sorted for a GROUP BY clause. Y=Yes; N=No. |
| TSLOCKMODE | An indication of the mode of lock to be acquired on either the new table, or its table space or table space partitions. If the isolation can be determined at bind time, the values are:<br>**IS** Intent share lock<br>**IX** Intent exclusive lock<br>**S** Share lock<br>**U** Update lock<br>**X** Exclusive lock<br>**SIX** Share with intent exclusive lock<br>**N** UR isolation; no lock<br>If the isolation cannot be determined at bind time, then the lock mode determined by the isolation at run time is shown by the following values.<br>**NS** For UR isolation, no lock; for CS, RS, or RR, an S lock.<br>**NIS** For UR isolation, no lock; for CS, RS, or RR, an IS lock.<br>**NSS** For UR isolation, no lock; for CS or RS, an IS lock; for RR, an S lock.<br>**SS** For UR, CS, or RS isolation, an IS lock; for RR, an S lock.<br>The data in this column is right justified. For example, IX appears as a blank followed by I followed by X. If the column contains a blank, then no lock is acquired. |
| TIMESTAMP | Usually, the time at which the row is processed, to the last .01 second. If necessary, DB2 adds .01 second to the value to ensure that rows for two successive queries have different values. |
| REMARKS | A field into which you can insert any character string of 254 or fewer characters. |
| PREFETCH | Whether data pages are to be read in advance by prefetch. S = pure sequential prefetch; L = prefetch through a page list; blank = unknown or no prefetch. |

*Table 111. Descriptions of columns in PLAN_TABLE (continued)*

| Column Name | Description |
|---|---|
| COLUMN_FN_EVAL | When an SQL column function is evaluated. R = while the data is being read from the table or index; S = while performing a sort to satisfy a GROUP BY clause; blank = after data retrieval and after any sorts. |
| MIXOPSEQ | The sequence number of a step in a multiple index operation.<br>**1, 2, ... n** For the steps of the multiple index procedure (ACCESSTYPE is MX, MI, or MU.)<br>**0** For any other rows (ACCESSTYPE is I, I1, M, N, R, or blank.) |
| VERSION | The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. Blank if not applicable. |
| COLLID | The collection ID for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. Blank if not applicable. |
| **Note:** The following nine columns, from ACCESS_DEGREE through CORRELATION_NAME, contain the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, each of them can contain null if the method it refers to does not apply. | |
| ACCESS_DEGREE | The number of parallel tasks or operations activated by a query. This value is determined at bind time; the actual number of parallel operations used at execution time could be different. This column contains 0 if there is a host variable. |
| ACCESS_PGROUP_ID | The identifier of the parallel group for accessing the new table. A parallel group is a set of consecutive operations, executed in parallel, that have the same number of parallel tasks. This value is determined at bind time; it could change at execution time. |
| JOIN_DEGREE | The number of parallel operations or tasks used in joining the composite table with the new table. This value is determined at bind time and can be 0 if there is a host variable. The actual number of parallel operations or tasks used at execution time could be different. |
| JOIN_PGROUP_ID | The identifier of the parallel group for joining the composite table with the new table. This value is determined at bind time; it could change at execution time. |
| SORTC_PGROUP_ID | The parallel group identifier for the parallel sort of the composite table. |
| SORTN_PGROUP_ID | The parallel group identifier for the parallel sort of the new table. |
| PARALLELISM_MODE | The kind of parallelism, if any, that is used at bind time:<br>**I** Query I/O parallelism<br>**C** Query CP parallelism<br>**X** Sysplex query parallelism |
| MERGE_JOIN_COLS | The number of columns that are joined during a merge scan join (Method=2). |
| CORRELATION_NAME | The correlation name of a table or view that is specified in the statement. If there is no correlation name, then the column is blank. |
| PAGE_RANGE | Whether the table qualifies for page range screening, so that plans scan only the partitions that are needed. Y = Yes; blank = No. |
| JOIN_TYPE | The type of join:<br>**F** FULL OUTER JOIN<br>**L** LEFT OUTER JOIN<br>**S** STAR JOIN<br>**blank** INNER JOIN or no join<br>RIGHT OUTER JOIN converts to a LEFT OUTER JOIN when you use it, so that JOIN_TYPE contains L. |
| GROUP_MEMBER | The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed. |
| IBM_SERVICE_DATA | Values are for IBM use only. |

*Table 111. Descriptions of columns in PLAN_TABLE (continued)*

| Column Name | Description |
|---|---|
| WHEN_OPTIMIZE | When the access path was determined:<br>**blank** At bind time, using a default filter factor for any host variables, parameter markers, or special registers.<br>**B** At bind time, using a default filter factor for any host variables, parameter markers, or special registers; however, the statement is reoptimized at run time using input variable values for input host variables, parameter markers, or special registers. The bind option REOPT(VARS) must be specified for reoptimization to occur.<br>**R** At run time, using input variables for any host variables, parameter markers, or special registers. The bind option REOPT(VARS) must be specified for this to occur. |
| QBLOCK_TYPE | For each query block, an indication of the type of SQL operation performed. For the outermost query, this column identifies the statement type. Possible values:<br>**SELECT** SELECT<br>**INSERT** INSERT<br>**UPDATE** UPDATE<br>**DELETE** DELETE<br>**SELUPD** SELECT with FOR UPDATE OF<br>**DELCUR** DELETE WHERE CURRENT OF CURSOR<br>**UPDCUR** UPDATE WHERE CURRENT OF CURSOR<br>**CORSUB** Correlated subquery<br>**NCOSUB** Noncorrelated subquery<br>**TABLEX** Table expression<br>**UNION** UNION<br>**UNIONA** UNION ALL |
| BIND_TIME | The time at which the plan or package for this statement or query block was bound. For static SQL statements, this is a full-precision timestamp value. For dynamic SQL statements, this is the value contained in the TIMESTAMP column of PLAN_TABLE appended by 4 zeroes. |
| OPTHINT | A string that you use to identify this row as an optimization hint for DB2. DB2 uses this row as input when choosing an access path. |
| HINT_USED | If DB2 used one of your optimization hints, it puts the identifier for that hint (the value in OPTHINT) in this column. |
| PRIMARY_ACCESSTYPE | Indicates whether direct row access will be attempted first:<br>**D** DB2 will try to use direct row access. If DB2 cannot use direct row access at runtime, it uses the access path described in the ACCESSTYPE column of PLAN_TABLE. See "Is direct row access possible? (PRIMARY_ACCESSTYPE = D)" on page 801 for more information.<br>**blank** DB2 will not try to use direct row access. |
| # PARENT_QBLOCKNO | A number that indicates the QBLOCKNO of the parent query block. |
| TABLE_TYPE | The type of new table:<br>**F** Table function<br>**Q** Temporary intermediate result table (not materialized)<br>**T** Table<br>**W** Work file<br>The value of the column is null if the query uses GROUP BY, ORDER BY, or DISTINCT, which requires an implicit sort. |

# Populating and maintaining a plan table

For the two distinct ways to populate a plan table, see:

- “Executing the SQL statement EXPLAIN”
- “Binding with the option EXPLAIN(YES)”

When you populate the plan table through DB2’s EXPLAIN, any INSERT triggers on the table are not activated. If you insert rows yourself, then those triggers are activated.

For a variation on the first way, see “Executing EXPLAIN under QMF”.

For tips on maintaining a growing plan table, see “Maintaining a plan table” on page 797.

## Executing the SQL statement EXPLAIN

You can populate PLAN_TABLE by executing the SQL statement EXPLAIN. In the statement, specify a single explainable SQL statement in the FOR clause.

You can execute EXPLAIN either statically from an application program, or dynamically, using QMF or SPUFI. For instructions and for details of the authorization you need on PLAN_TABLE, see *DB2 SQL Reference*.

## Binding with the option EXPLAIN(YES)

You can populate a plan table when you bind or rebind a plan or package. Specify the option EXPLAIN(YES). EXPLAIN obtains information about the access paths for all explainable SQL statements in a package or the DBRMs of a plan. The information appears in table *package_owner*.PLAN_TABLE or *plan_owner*.PLAN_TABLE. For dynamically prepared SQL, the qualifier of PLAN_TABLE is the current SQLID.

***Performance considerations:*** EXPLAIN as a bind option should not be a performance concern. The same processing for access path selection is performed, regardless of whether you use EXPLAIN(YES) or EXPLAIN (NO). With EXPLAIN(YES), there is only a small amount of overhead processing to put the results in a plan table.

If a plan or package that was previously bound with EXPLAIN(YES) is automatically rebound, the value of field EXPLAIN PROCESSING on installation panel DSNTIPO determines whether EXPLAIN is run again during the automatic rebind. Again, there is a small amount of overhead for inserting the results into a plan table.

***EXPLAIN for remote binds:*** A remote requester that accesses DB2 can specify EXPLAIN(YES) when binding a package at the DB2 server. The information appears in a plan table at the server, not at the requester. If the requester does not support the propagation of the option EXPLAIN(YES), rebind the package at the requester with that option to obtain access path information. You cannot get information about access paths for SQL statements that use private protocol.

## Executing EXPLAIN under QMF

You can use QMF to display the results of EXPLAIN to the terminal. You can create your own form to display the output or use QMF’s default form.

***Use parameter markers for host variables:*** If you have host variables in a predicate for an original query in a static application and if you are using QMF or SPUFI to execute EXPLAIN for the query, in most cases, use parameter markers where you use host variables in the original query. If you a literal value instead, you might see different access paths for your static and dynamic queries. For instance, compare the following queries:

```
                        QMF Query Using Parameter   QMF Query Using
Original Static SQL       Marker                       Literal

DECLARE C1               EXPLAIN PLAN SET            EXPLAIN PLAN SET
CURSOR FOR               QUERYNO=1 FOR               QUERYNO=1 FOR
SELECT *                 SELECT *                    SELECT *
 FROM T1                  FROM T1                    FROM T1
 WHERE C1 > HOST VAR.     WHERE C1 > ?               WHERE C1 > 10
```

Using the literal ’10’ would likely produce a different filter factor and maybe a different access path from the original static SQL. (A filter factor is the proportion of rows that remain after a predicate has ″filtered out″ the rows that do not satisfy it. For more information on filter factors, see “Predicate filter factors” on page 723.) The parameter marker behaves just like a host variable, in that the predicate is assigned a default filter factor.

***When to use a literal:*** If you know that the static plan or package was bound with REOPT(VARS) and you have some idea of what is returned in the host variable, it can be more accurate to include the literal in the QMF EXPLAIN. REOPT(VARS) means that DB2 will replace the value of the host variable with the true value at run time and then determine the access path. For more information about REOPT(VARS) see “Using REOPT(VARS) to change the access path at run time” on page 734.

***Expect these differences:*** Even when using parameter markers, you could see different access paths for static and dynamic queries. DB2 assumes that the value that replaces a parameter marker has the same length and precision as the column it is compared to. That assumption determines whether the predicate is indexable or stage 1. However, if a host variable definition does not match the column definition, then the predicate may become a stage 2 predicate and, hence, nonindexable.

The host variable definition fails to match the column definition if:

- The length of the host variable is greater than the length attribute of the column.
- The precision of the host variable is greater than that of the column.
- The data type of the host variable is not compatible with the data type of the column. For example, you cannot use a host variable with data type DECIMAL with a column of data type SMALLINT. But you can use a host variable with data type SMALLINT with a column of data type INT or DECIMAL.

### Maintaining a plan table

DB2 adds rows to PLAN_TABLE as you choose; it does not automatically delete rows. To clear the table of obsolete rows, use DELETE, just as you would for deleting rows from any table. You can also use DROP TABLE to drop a plan table completely.

## Reordering rows from a plan table

Several processes can insert rows into the same plan table. To understand access paths, you must retrieve the rows for a particular query in an appropriate order.

### Retrieving rows for a plan

The rows for a particular plan are identified by the value of APPLNAME. The following query to a plan table returns the rows for all the explainable statements in a plan in their logical order:

```
SELECT * FROM JOE.PLAN_TABLE
   WHERE APPLNAME = 'APPL1'
   ORDER BY TIMESTAMP, QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

The result of the ORDER BY clause shows whether there are:

- Multiple QBLOCKNOs within a QUERYNO
- Multiple PLANNOs within a QBLOCKNO
- Multiple MIXOPSEQs within a PLANNO

All rows with the same non-zero value for QBLOCKNO and the same value for QUERYNO relate to a step within the query. QBLOCKNOs are not necessarily executed in the order shown in PLAN_TABLE. But within a QBLOCKNO, the PLANNO column gives the substeps in the order they execute.

For each substep, the TNAME column identifies the table accessed. Sorts can be shown as part of a table access or as a separate step.

***What if QUERYNO=0?*** In a program with more than 32767 lines, all values of QUERYNO greater than 32767 are reported as 0. For entries containing QUERYNO=0, use the timestamp, which is guaranteed to be unique, to distinguish individual statements.

### Retrieving rows for a package

The rows for a particular package are identified by the values of PROGNAME, COLLID, and VERSION. Those columns correspond to the following four-part naming convention for packages:

```
LOCATION.COLLECTION.PACKAGE_ID.VERSION
```

COLLID gives the COLLECTION name, and PROGNAME gives the PACKAGE_ID. The following query to a plan table return the rows for all the explainable statements in a package in their logical order:

```
SELECT * FROM JOE.PLAN_TABLE
   WHERE PROGNAME = 'PACK1' AND COLLID = 'COLL1' AND VERSION =  'PROD1'
   ORDER BY QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

## Asking questions about data access

When you examine your EXPLAIN results, try to answer the following questions:

- “Is access through an index? (ACCESSTYPE is I, I1, N or MX)” on page 799
- “Is access through more than one index? (ACCESSTYPE=M)” on page 799
- “How many columns of the index are used in matching? (MATCHCOLS=n)” on page 800
- “Is the query satisfied using only the index? (INDEXONLY=Y)” on page 800
- “Is direct row access possible? (PRIMARY_ACCESSTYPE = D)” on page 801
- “Is a view or nested table expression materialized?” on page 803
- “Was a scan limited to certain partitions? (PAGE_RANGE=Y)” on page 803
- “What kind of prefetching is done? (PREFETCH = L, S, or blank)” on page 803
- “Is data accessed or processed in parallel? (PARALLELISM_MODE is I, C, or X)” on page 804
- “Are sorts performed?” on page 804
- “Is a subquery transformed into a join?” on page 805
- “When are column functions evaluated? (COLUMN_FN_EVAL)” on page 805

As explained in this section, they can be answered in terms of values in columns of a plan table.

## Is access through an index? (ACCESSTYPE is I, I1, N or MX)

If the column ACCESSTYPE in the plan table has one of those values, DB2 uses an index to access the table named in column TNAME. The columns ACCESSCREATOR and ACCESSNAME identify the index. For a description of methods of using indexes, see “Index access paths” on page 807.

## Is access through more than one index? (ACCESSTYPE=M)

Those values indicate that DB2 uses a set of indexes to access a single table. A set of rows in the plan table contain information about the multiple index access. The rows are numbered in column MIXOPSEQ in the order of execution of steps in the multiple index access. (If you retrieve the rows in order by MIXOPSEQ, the result is similar to postfix arithmetic notation.)

The examples in Figure 100 and Figure 101 on page 800 have these indexes: IX1 on T(C1) and IX2 on T(C2). DB2 processes the query in the following steps:

1. Retrieve all the qualifying record identifiers (RIDs) where C1=1, using index IX1.
2. Retrieve all the qualifying RIDs where C2=1, using index IX2. The intersection of those lists is the final set of RIDs.
3. Access the data pages needed to retrieve the qualified rows using the final RID list.

```
SELECT * FROM T
  WHERE C1 = 1 AND C2 = 1;
```

| TNAME | ACCESS-TYPE | MATCH-COLS | ACCESS-NAME | INDEX-ONLY | PREFETCH | MIXOP-SEQ |
|---|---|---|---|---|---|---|
| T | M | 0 | | N | L | 0 |
| T | MX | 1 | IX1 | Y | | 1 |
| T | MX | 1 | IX2 | Y | | 2 |
| T | MI | 0 | | N | | 3 |

*Figure 100. PLAN_TABLE output for example with intersection (AND) operator*

The same index can be used more than once in a multiple index access, because more than one predicate could be matching, as in Figure 101 on page 800.

```
SELECT * FROM T
  WHERE C1 BETWEEN 100 AND 199 OR
        C1 BETWEEN 500 AND 599;
```

| TNAME | ACCESS-TYPE | MATCH-COLS | ACCESS-NAME | INDEX-ONLY | PREFETCH | MIXOP-SEQ |
|---|---|---|---|---|---|---|
| T | M | 0 |  | N | L | 0 |
| T | MX | 1 | IX1 | Y |  | 1 |
| T | MX | 1 | IX1 | Y |  | 2 |
| T | MU | 0 |  | N |  | 3 |

*Figure 101. PLAN_TABLE output for example with union (OR) operator*

DB2 processes the query in the following steps:

1. Retrieve all RIDs where C1 is between 100 and 199, using index IX1.
2. Retrieve all RIDs where C1 is between 500 and 599, again using IX1. The union of those lists is the final set of RIDs.
3. Retrieve the qualified rows using the final RID list.

## How many columns of the index are used in matching? (MATCHCOLS=n)

If MATCHCOLS is 0, the access method is called a *nonmatching index scan*. All the index keys and their RIDs are read.

If MATCHCOLS is greater than 0, the access method is called a *matching index scan*: the query uses predicates that match the index columns.

In general, the matching predicates on the leading index columns are equal or IN predicates. The predicate that matches the final index column can be an equal, IN, or range predicate (<, <=, >, >=, LIKE, or BETWEEN).

The following example illustrates matching predicates:

```
SELECT * FROM EMP
  WHERE JOBCODE = '5' AND SALARY > 60000 AND LOCATION = 'CA';

 INDEX XEMP5 on (JOBCODE, LOCATION, SALARY, AGE);
```

The index XEMP5 is the chosen access path for this query, with MATCHCOLS = 3. Two equal predicates are on the first two columns and a range predicate is on the third column. Though the index has four columns in the index, only three of them can be considered matching columns.

## Is the query satisfied using only the index? (INDEXONLY=Y)

In this case, the method is called *index-only access*. For a SELECT operation, all the columns needed for the query can be found in the index and DB2 does not access the table. For an UPDATE or DELETE operation, only the index is required to read the selected row.

Index-only access to data is not possible for any step that uses list prefetch, which is described under “What kind of prefetching is done? (PREFETCH = L, S, or blank)” on page 803. Index-only access is not possible when returning varying-length data in the result set or a VARCHAR column has a LIKE predicate, unless the VARCHAR FROM INDEX field of installation panel DSNTIP4 is set to

YES and plan or packages have been rebound to pick up the change. See Part 2 of *DB2 Installation Guide* for more information.

If access is by more than one index, INDEXONLY is Y for a step with access type MX, because the data pages are not actually accessed until all the steps for intersection (MI) or union (MU) take place.

When an SQL application uses index-only access for a ROWID column, the application claims the table space or table space partition. As a result, contention may occur between the SQL application and a utility that drains the table space or partition. Index-only access to a table for a ROWID column is not possible if the associated table space or partition is in an incompatible restrictive state. For example, an SQL application can make a read claim on the table space only if the restrictive state allows readers.

## Is direct row access possible? (PRIMARY_ACCESSTYPE = D)

If an application selects a row from a table that contains a ROWID column, the row ID value implicitly contains the location of the row. If you use that row ID value in the search condition of subsequent SELECTs, DELETEs, or UPDATEs, DB2 might be able to navigate directly to the row. This access method is called *direct row access*.

Direct row access is very fast, because DB2 does not need to use the index or a table space scan to find the row. Direct row access can be used on any table that has a ROWID column.

To use direct row access, you first select the values of a row into host variables. The value that is selected from the ROWID column contains the location of that row. Later, when you perform queries that access that row, you include the row ID value in the search condition. If DB2 determines that it can use direct row access, it uses the row ID value to navigate directly to the row.

### Which predicates qualify for direct row access?

For a query to qualify for direct row access, the search condition must be a Boolean term *stage 1* predicate that fits one of these descriptions:

1. A simple Boolean term predicate of the form COL=*noncolumn expression*, where COL has the ROWID data type and *noncolumn expression* contains a row ID
2. A simple Boolean term predicate of the form COL IN *list*, where COL has the ROWID data type and the values in *list* are row IDs, and an index is defined on COL
3. A compound Boolean term that combines several simple predicates using the AND operator, and one of the simple predicates fits description 1 or 2

However, just because a query qualifies for direct row access does not mean that that access path is always chosen. If DB2 determines that another access path is better, direct row access is not chosen.

***Examples:*** In the following predicate example, ID is a ROWID column in table T1. A unique index exists on that ID column. The host variables are of the ROWID type.

```
WHERE ID IN (:hv_rowid1,:hv_rowid2,:hv_rowid3)
```

The following predicate also qualifies for direct row access:

```
WHERE ID = ROWID(X'F0DFD230E3C0D80D81C201AA0A280100000000000203')
```

***Searching for propagated rows:*** If rows are propagated from one table to another, do not expect to use the same row ID value from the source table to search for the same row in the target table, or vice versa. This does not work when direct row access is the access path chosen. For example, assume that the host variable below contains a row ID from SOURCE:

```
SELECT * FROM TARGET
  WHERE ID = :hv_rowid
```

Because the row ID location is not the same as in the source table, DB2 will most likely not find that row. Search on another column to retrieve the row you want.

## Reverting to ACCESSTYPE

Although DB2 might plan to use direct row access, circumstances can cause DB2 to not use direct row access at run time. DB2 remembers the location of the row as of the time it is accessed. However, that row can change locations (such as after a REORG) between the first and second time it is accessed, which means that DB2 cannot use direct row access to find the row on the second access attempt. Instead of using direct row access, DB2 uses the access path that is shown in the ACCESSTYPE column of PLAN_TABLE.

If the predicate you are using to do direct row access is not indexable and if DB2 is unable to use direct row access, then DB2 uses a table space scan to find the row. This can have a profound impact on the performance of applications that rely on direct row access. Write your applications to handle the possibility that direct row access might not be used. Some options are to:

- Ensure that your application does not try to remember ROWID columns across reorganizations of the table space.

  When your application commits, it releases its claim on the table space; it is possible that a REORG can run and move the row, which disables direct row access. Plan your commit processing accordingly; use the returned row ID value before committing, or re-select the row ID value after a commit is issued.

  If you are storing ROWID columns from another table, update those values after the table with the ROWID column is reorganized.
- Create an index on the ROWID column, so that DB2 can use the index if direct row access is disabled.
- Supplement the ROWID column predicate with another predicate that enables DB2 to use an existing index on the table. For example, after reading a row, an application might perform the following update:

  ```
  EXEC SQL UPDATE EMP
  SET SALARY = :hv_salary + 1200
   WHERE EMP_ROWID = :hv_emp_rowid
   AND EMPNO = :hv_empno;
  ```

  If an index exists on EMPNO, DB2 can use index access if direct access fails. The additional predicate ensures DB2 does not revert to a table space scan.

## Using direct row access and other access methods

***Parallelism:*** Direct row access and parallelism are mutually exclusive. If a query qualifies for both direct row access and parallelism, direct row access is used. If direct row access fails, DB2 does not revert to parallelism; instead it reverts to the backup access type (as designated by column ACCESSTYPE in the PLAN_TABLE). This might result in a table space scan. To avoid a table space scan in case direct row access fails, add an indexed column to the predicate.

***RID list processing:*** Direct row access and RID list processing are mutually exclusive. If a query qualifies for both direct row access and RID list processing, direct row access is used. If direct row access fails, DB2 does not revert to RID list processing; instead it reverts to the backup access type.

## Is a view or nested table expression materialized?

When the column TNAME names a view or nested table expression and column TABLE_TYPE contains a W, it indicates that the view or nested table expresssion is materialized. *Materialization* means that the data rows selected by the view or nested table expression are put into a work file to be processed like a table. (For a more detailed description of materialization, see “Processing for views and nested table expressions” on page 829.)

## Was a scan limited to certain partitions? (PAGE_RANGE=Y)

DB2 can limit a scan of data in a partitioned table space to one or more partitions. The method is called a *limited partition scan*. The query must provide a predicate on the first key column of the partitioning index. Only the first key column is significant for limiting the range of the partition scan.

A limited partition scan can be combined with other access methods. For example, consider the following query:

```
SELECT .. FROM T
   WHERE (C1 BETWEEN '2002' AND '3280'
   OR C1 BETWEEN '6000' AND '8000')
   AND C2 = '6';
```

Assume that table T has a partitioned index on column C1 and that values of C1 between 2002 and 3280 all appear in partitions 3 and 4 and the values between 6000 and 8000 appear in partitions 8 and 9. Assume also that T has another index on column C2. DB2 could choose any of these access methods:

- A matching index scan on column C1. The scan reads index values and data only from partitions 3, 4, 8, and 9. (PAGE_RANGE=N)
- A matching index scan on column C2. (DB2 might choose that if few rows have C2=6.) The matching index scan reads all RIDs for C2=6 from the index on C2 and corresponding data pages from partitions 3, 4, 8, and 9. (PAGE_RANGE=Y)
- A table space scan on T. DB2 avoids reading data pages from any partitions except 3, 4, 8 and 9. (PAGE_RANGE=Y)

***Joins:*** Limited partition scan can be used for each table accessed in a join.

***Restrictions:*** Limited partition scan is not supported when host variables or parameter markers are used on the first key of the primary index. This is because the qualified partition range based on such a predicate is unknown at bind time. If you think you can benefit from limited partition scan but you have host variables or parameter markers, consider binding with REOPT(VARS).

If you have predicates using an OR operator and one of the predicates refers to a column of the partitioning index that is not the first key column of the index, then DB2 does not use limited partition scan.

## What kind of prefetching is done? (PREFETCH = L, S, or blank)

*Prefetching* is a method of determining in advance that a set of data pages is about to be used and then reading the entire set into a buffer with a single asynchronous I/O operation. If the value of PREFETCH is:

- S, the method is called *sequential prefetch*. The data pages that are read in advance are sequential. A table space scan always uses sequential prefetch. An index scan might not use it. For a more complete description, see “Sequential prefetch (PREFETCH=S)” on page 824.
- L, the method is called *list prefetch*. One or more indexes are used to select the RIDs for a list of data pages to be read in advance; the pages need not be sequential. Usually, the RIDs are sorted. The exception is the case of a hybrid join (described under “Hybrid join (METHOD=4)” on page 818) when the value of column SORTN_JOIN is N. For a more complete description, see “List prefetch (PREFETCH=L)” on page 825.
- Blank, prefetching is not chosen as an access method. However, depending on the pattern of the page access, data can be prefetched at execution time through a process called *sequential detection*. For a description of that process, see “Sequential detection at execution time” on page 826.

## Is data accessed or processed in parallel? (PARALLELISM_MODE is I, C, or X)

Parallel processing applies only to read-only queries.

| **If mode is:** | **DB2 plans to use:** |
|---|---|
| **I** | Parallel I/O operations |
| **C** | Parallel CP operations |
| **X** | Sysplex query parallelism |

Non-null values in columns ACCESS_DEGREE and JOIN_DEGREE indicate to what degree DB2 plans to use parallel operations. At execution time, however, DB2 might not actually use parallelism, or it might use fewer operations in parallel than were originally planned. For a more complete description , see “Chapter 34. Parallel operations and query performance” on page 841. For more information about Sysplex query parallelism, see Chapter 6 of *DB2 Data Sharing: Planning and Administration*.

## Are sorts performed?

***SORTN_JOIN and SORTC_JOIN:*** SORTN_JOIN indicates that the new table of a join is sorted before the join. (For hybrid join, this is a sort of the RID list.) When SORTN_JOIN and SORTC_JOIN are both ’Y’, two sorts are performed for the join. The sorts for joins are indicated on the same row as the new table access.

***METHOD 3 sorts:*** These are used for ORDER BY, GROUP BY, SELECT DISTINCT, UNION, or a quantified predicate. A quantified predicate is ’col = ANY (fullselect)’ or ’col = SOME (fullselect)’ . They are indicated on a separate row. A single row of the plan table can indicate two sorts of a composite table, but only one sort is actually done.

***SORTC_UNIQ and SORTC_ORDERBY:*** SORTC_UNIQ indicates a sort to remove duplicates, as might be needed by a SELECT statement with DISTINCT or UNION. SORTC_ORDERBY usually indicates a sort for an ORDER BY clause. But SORTC_UNIQ and SORTC_ORDERBY also indicate when the results of a noncorrelated subquery are sorted, both to remove duplicates and to order the results. One sort does both the removal and the ordering.

## Is a subquery transformed into a join?

For better access paths, DB2 sometimes transforms subqueries into joins, as described in "Subquery transformation into join" on page 741. A plan table shows that a subquery is transformed into a join by the value in column QBLOCKNO.

- If the subquery is not transformed into a join, that means it is executed in a separate operation, and its value of QBLOCKNO is greater than the value for the outer query.
- If the subquery is transformed into a join, it and the outer query have the same value of QBLOCKNO. A join is also indicated by a value of 1, 2, or 4 in column METHOD.

## When are column functions evaluated? (COLUMN_FN_EVAL)

When the column functions are evaluated is based on the access path chosen for the SQL statement.

- If the ACCESSTYPE column is I1, then a MAX or MIN function can be evaluated by one access of the index named in ACCESSNAME.
- For other values of ACCESSTYPE, the COLUMN_FN_EVAL column tells when DB2 is evaluating the column functions.

| **Value** | **Functions are evaluated ...** |
|---|---|
| **S** | While performing a sort to satisfy a GROUP BY clause |
| **R** | While the data is being read from the table or index |
| **blank** | After data retrieval and after any sorts |

Generally, values of R and S are considered better for performance than a blank.

***Use variance and standard deviation with care:*** The VARIANCE and STDDEV functions are always evaluated late (that is, COLUMN_FN_EVAL is blank). This causes other functions in the same query block to be evaluated late as well. For example, in the following query, the sum function is evaluated later than it would be if the variance function was not present:

```
SELECT SUM(C1), VARIANCE(C1) FROM T1;
```

## Interpreting access to a single table

The following sections describe different access paths that values in a plan table can indicate, along with suggestions for supplying better access paths for DB2 to choose from.

- Table space scans (ACCESSTYPE=R PREFETCH=S)
- "Overview of index access" on page 806
- "Index access paths" on page 807
- "UPDATE using an index" on page 812

## Table space scans (ACCESSTYPE=R PREFETCH=S)

Table space scan is most often used for one of the following reasons:

- Access is through a created temporary table. (Index access is not possible for created temporary tables.)
- A matching index scan is not possible because an index is not available, or no predicates match the index columns.
- A high percentage of the rows in the table is returned. In this case, an index is not really useful because most rows need to be read anyway.
- The indexes that have matching predicates have low cluster ratios and are therefore efficient only for small amounts of data.

Assume that table T has no index on C1. The following is an example that uses a table space scan:

```
SELECT * FROM T WHERE C1 = VALUE;
```

In this case, at least every row in T must be examined to determine whether the value of C1 matches the given value.

### Table space scans of nonsegmented table spaces

DB2 reads and examines every page in the table space, regardless of which table the page belongs to. It might also read pages that have been left as free space and space not yet reclaimed after deleting data.

### Table space scans of segmented table spaces

If the table space is segmented, DB2 first determines which segments need to be read. It then reads only the segments in the table space that contain rows of T. If the prefetch quantity, which is determined by the size of your buffer pool, is greater than the SEGSIZE and if the segments for T are not contiguous, DB2 might read unnecessary pages. Use a SEGSIZE value that is as large as possible, consistent with the size of the data. A large SEGSIZE value is best to maintain clustering of data rows. For very small tables, specify a SEGSIZE value that is equal to the number of pages required for the table.

***Recommendation for SEGSIZE value:*** Table 112 summarizes the recommendations for SEGSIZE, depending on how large the table is.

*Table 112. Recommendations for SEGSIZE*

| Number of pages | SEGSIZE recommendation |
|---|---|
| ≤ 28 | 4 to 28 |
| > 28 < 128 pages | 32 |
| ≥ 128 pages | 64 |

### Table space scans of partitioned table spaces

Partitioned table spaces are nonsegmented. A table space scan on a partitioned table space is more efficient than on a nonpartitioned table space. DB2 takes advantage of the partitions by a limited partition scan, as described under “Was a scan limited to certain partitions? (PAGE_RANGE=Y)” on page 803.

### Table space scans and sequential prefetch

Regardless of the type of table space, DB2 plans to use sequential prefetch for a table space scan. For a segmented table space, DB2 might not actually use sequential prefetch at execution time if it can determine that fewer than four data pages need to be accessed. For guidance on monitoring sequential prefetch, see “Sequential prefetch (PREFETCH=S)” on page 824.

If you do not want to use sequential prefetch for a particular query, consider adding to it the clause OPTIMIZE FOR 1 ROW.

## Overview of index access

Indexes can provide efficient access to data. In fact, that is the only purpose of nonunique indexes. Unique indexes have the additional purpose of ensuring that key values are unique.

### Using indexes to avoid sorts

As well as providing selective access to data, indexes can also order data, sometimes eliminating the need for sorting. Some sorts can be avoided if index keys are in the order needed by ORDER BY, GROUP BY, a join operation, or DISTINCT in a column function. In other cases, as when list prefetch is used, the index does not provide useful ordering, and the selected data might have to be sorted.

When it is absolutely necessary to prevent a sort, consider creating an index on the column or columns necessary to provide that ordering. Consider also using the clause OPTIMIZE FOR 1 ROW to discourage DB2 from choosing a sort for the access path.

Consider the following query:

```
SELECT C1,C2,C3 FROM T
  WHERE C1 > 1
    ORDER BY C1 OPTIMIZE FOR 1 ROW;
```

An ascending index on C1 or an index on (C1,C2,C3) could eliminate a sort. (For more information on OPTIMIZE FOR n ROWS, see “Minimizing overhead for retrieving few rows: OPTIMIZE FOR n ROWS” on page 747.)

Not all sorts are inefficient. For example, if the index that provides ordering is not an efficient one and many rows qualify, it is possible that using another access path to retrieve and then sort the data could be more efficient than the inefficient, ordering index.

### Costs of indexes

Before you begin creating indexes, consider carefully their costs:

- Indexes require storage space.
- Each index requires an index space and a data set, and operating system restrictions exist on the number of open data sets.
- Indexes must be changed to reflect every insert or delete operation on the base table. If an update operation updates a column that is in the index, then the index must also be changed. The time required by these operations increases accordingly.
- Indexes can be built automatically when loading data, but this takes time. They must be recovered or rebuilt if the underlying table space is recovered, which might also be time-consuming.

***Recommendation:*** In reviewing the access paths described in the next section, consider indexes as part of your database design, See “Part 2. Designing a database: advanced topics” on page 27 for details about database design in general. For a query with a performance problem, ask yourself:

- Would adding a column to an index allow the query to use index-only access?
- Do you need a new index?
- Is your choice of clustering index correct?

## Index access paths

DB2 uses the following index access paths:

- “Matching index scan (MATCHCOLS>0)” on page 808
- “Index screening” on page 808
- “Nonmatching index scan (ACCESSTYPE=I and MATCHCOLS=0)” on page 809
- “IN-list index scan (ACCESSTYPE=N)” on page 809
- “Multiple index access (ACCESSTYPE is M, MX, MI, or MU)” on page 809
- “One-fetch access (ACCESSTYPE=I1)” on page 811

- “Index-only access (INDEXONLY=Y)” on page 811
- “Equal unique index (MATCHCOLS=number of index columns)” on page 811

## Matching index scan (MATCHCOLS>0)

In a *matching index scan*, predicates are specified on either the leading or all of the index key columns. These predicates provide *filtering*; only specific index pages and data pages need to be accessed. If the degree of filtering is high, the matching index scan is efficient.

In the general case, the rules for determining the number of matching columns are simple, although there are a few exceptions.

- Look at the index columns from leading to trailing. For each index column, search for an indexable boolean term predicate on that column. (See “Properties of predicates” on page 714 for a definition of boolean term.) If such a predicate is found, then it can be used as a matching predicate.

  Column MATCHCOLS in a plan table shows how many of the index columns are matched by predicates.
- If no matching predicate is found for a column, the search for matching predicates stops.
- If a matching predicate is a range predicate, then there can be no more matching columns. For example, in the matching index scan example that follows, the range predicate `C2>1` prevents the search for additional matching columns.
- For star joins, a missing key predicate does not cause termination of matching columns that are to be used on the fact table index.

The exceptional cases are:

- At most one IN-list predicate can be a matching predicate on an index.
- For MX accesses and index access with list prefetch, IN-list predicates cannot be used as matching predicates.
- Join predicates cannot qualify as matching predicates when doing a merge join (METHOD=2). For example, T1.C1=T2.C1 cannot be a matching predicate when doing a merge join, although any local predicates, such as C1=’5’ can be used.

  Join predicates can be used as matching predicates on the inner table of a nested loop join or hybrid join.

***Matching index scan example:*** Assume there is an index on T(C1,C2,C3,C4):

```
SELECT * FROM T
  WHERE C1=1 AND C2>1
  AND C3=1;
```

Two matching columns occur in this example. The first one comes from the predicate C1=1, and the second one comes from C2>1. The range predicate on C2 prevents C3 from becoming a matching column.

## Index screening

In *index screening*, predicates are specified on index key columns but are not part of the matching columns. Those predicates improve the index access by reducing the number of rows that qualify while searching the index. For example, with an index on T(C1,C2,C3,C4) in the following SQL statement, C3>0 and C4=2 are index screening predicates.

```
SELECT * FROM T
  WHERE C1 = 1
   AND C3 > 0 AND C4 = 2
   AND C5 = 8;
```

The predicates can be applied on the index, but they are not matching predicates. C5=8 is not an index screening predicate, and it must be evaluated when data is retrieved. The value of MATCHCOLS in the plan table is 1.

EXPLAIN does not directly tell when an index is screened; however, if MATCHCOLS is less than the number of index key columns, it indicates that index screening is possible.

## Nonmatching index scan (ACCESSTYPE=I and MATCHCOLS=0)

In a *nonmatching index scan* no matching columns are in the index. Hence, all the index keys must be examined.

Because a nonmatching index usually provides no filtering, only a few cases provide an efficient access path. The following situations are examples:

- When index screening predicates exist

  In that case, not all of the data pages are accessed.
- When the clause OPTIMIZE FOR n ROWS is used

  That clause can sometimes favor a nonmatching index, especially if the index gives the ordering of the ORDER BY clause.
- When more than one table exists in a nonsegmented table space

  In that case, a table space scan reads irrelevant rows. By accessing the rows through the nonmatching index, fewer rows are read.

## IN-list index scan (ACCESSTYPE=N)

An *IN-list index scan* is a special case of the matching index scan, in which a single indexable IN predicate is used as a matching equal predicate.

You can regard the IN-list index scan as a series of matching index scans with the values in the IN predicate being used for each matching index scan. The following example has an index on (C1,C2,C3,C4) and might use an IN-list index scan:

```
SELECT * FROM T
  WHERE C1=1 AND C2 IN (1,2,3)
   AND C3>0 AND C4<100;
```

The plan table shows MATCHCOLS = 3 and ACCESSTYPE = N. The IN-list scan is performed as the following three matching index scans:

```
 (C1=1,C2=1,C3>0), (C1=1,C2=2,C3>0), (C1=1,C2=3,C3>0)
```

| Parallelism is supported for queries that involve IN-list index access. These queries used to run sequentially in previous releases of DB2, although parallelism could have been used when the IN-list access was for the inner table of a parallel group. Now, in environments in which parallelism is enabled, you can see a reduction in elapsed time for queries that involve IN-list index access for the outer table of a parallel group.

## Multiple index access (ACCESSTYPE is M, MX, MI, or MU)

*Multiple index access* uses more than one index to access a table. It is a good access path when:

- No single index provides efficient access.
- A combination of index accesses provides efficient access.

RID lists are constructed for each of the indexes involved. The unions or intersections of the RID lists produce a final list of qualified RIDs that is used to retrieve the result rows, using list prefetch. You can consider multiple index access as an extension to list prefetch with more complex RID retrieval operations in its first phase. The complex operators are union and intersection.

DB2 chooses multiple index access for the following query:

```
SELECT * FROM EMP
    WHERE (AGE = 34) OR
          (AGE = 40 AND JOB = 'MANAGER');
```

For this query:

- EMP is a table with columns EMPNO, EMPNAME, DEPT, JOB, AGE, and SAL.
- EMPX1 is an index on EMP with key column AGE.
- EMPX2 is an index on EMP with key column JOB.

The plan table contains a sequence of rows describing the access. For this query, ACCESSTYPE uses the following values:

| **Value** | **Meaning** |
|---|---|
| **M** | Start of multiple index access processing |
| **MX** | Indexes are to be scanned for later union or intersection |
| **MI** | An intersection (AND) is performed |
| **MU** | A union (OR) is performed |

The following steps relate to the previous query and the values shown for the plan table in Figure 102:

1. Index EMPX1, with matching predicate AGE= 34, provides a set of candidates for the result of the query. The value of MIXOPSEQ is 1.
2. Index EMPX1, with matching predicate AGE = 40, also provides a set of candidates for the result of the query. The value of MIXOPSEQ is 2.
3. Index EMPX2, with matching predicate JOB='MANAGER', also provides a set of candidates for the result of the query. The value of MIXOPSEQ is 3.
4. The first intersection (AND) is done, and the value of MIXOPSEQ is 4. This MI removes the two previous candidate lists (produced by MIXOPSEQs 2 and 3) by intersecting them to form an intermediate candidate list, IR1, which is not shown in PLAN_TABLE.
5. The last step, where the value MIXOPSEQ is 5, is a union (OR) of the two remaining candidate lists, which are IR1 and the candidate list produced by MIXOPSEQ 1. This final union gives the result for the query.

| **PLAN-NO** | **TNAME** | **ACCESS-TYPE** | **MATCH-COLS** | **ACCESS-NAME** | **PREFETCH** | **MIXOP-SEQ** |
|---|---|---|---|---|---|---|
| 1 | EMP | M | 0 |  | L | 0 |
| 1 | EMP | MX | 1 | EMPX1 |  | 1 |
| 1 | EMP | MX | 1 | EMPX1 |  | 2 |
| 1 | EMP | MI | 0 |  |  | 3 |
| 1 | EMP | MX | 1 | EMPX2 |  | 4 |
| 1 | EMP | MU | 0 |  |  | 5 |

*Figure 102. Plan table output for a query that uses multiple indexes. Depending on the filter factors of the predicates, the access steps can appear in a different order.*

In this example, the steps in the multiple index access follow the physical sequence of the predicates in the query. This is not always the case. The multiple index steps are arranged in an order that uses RID pool storage most efficiently and for the least amount of time.

## One-fetch access (ACCESSTYPE=I1)

*One-fetch index access* requires retrieving only one row. It is the best possible access path and is chosen whenever it is available. It applies to a statement with a MIN or MAX column function: the order of the index allows a single row to give the result of the function.

One-fetch index access is a possible access path when:

- There is only one table in the query.
- There is only one column function (either MIN or MAX).
- Either no predicate or all predicates are matching predicates for the index.
- There is no GROUP BY.
- Column functions are on:
  - The first index column if there are no predicates
  - The last matching column of the index if the last matching predicate is a range type
  - The next index column (after the last matching column) if all matching predicates are equal type

***Queries using one-fetch index access:*** Assuming that an index exists on `T(C1,C2,C3)`, the following queries use one-fetch index scan:

```
SELECT MIN(C1) FROM T;
SELECT MIN(C1) FROM T WHERE C1>5;
SELECT MIN(C1) FROM T WHERE C1>5 AND C1<10;
SELECT MIN(C2) FROM T WHERE C1=5;
SELECT MAX(C1) FROM T;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2<10;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2>5 AND C2<10;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2 BETWEEN 5 AND 10;
```

## Index-only access (INDEXONLY=Y)

With *index-only access*, the access path does not require any data pages because the access information is available in the index. Conversely, when an SQL statement requests a column that is not in the index, updates any column in the table, or deletes a row, DB2 has to access the associated data pages. Because the index is almost always smaller than the table itself, an index-only access path usually processes the data efficiently.

With an index on T(C1,C2), the following queries can use index-only access:

```
SELECT C1, C2 FROM T WHERE C1 > 0;
SELECT C1, C2 FROM T;
SELECT COUNT(*) FROM T WHERE C1 = 1;
```

## Equal unique index (MATCHCOLS=number of index columns)

An index that is fully matched and unique, and in which all matching predicates are equal-predicates, is called an *equal unique index* case. This case guarantees that only one row is retrieved. If there is no one-fetch index access available, this is considered the most efficient access over all other indexes that are not equal unique. (The uniqueness of an index is determined by whether or not it was defined as unique.)

Sometimes DB2 can determine that an index that is not fully matching is actually an equal unique index case. Assume the following case:

```
Unique Index1: (C1, C2)
Unique Index2: (C2, C1, C3)

SELECT C3 FROM T
 WHERE C1 = 1 AND
       C2 = 5;
```

Index1 is a fully matching equal unique index. However, Index2 is also an equal unique index even though it is not fully matching. Index2 is the better choice because, in addition to being equal and unique, it also provides index-only access.

## UPDATE using an index

If no index key columns are updated, you can use an index while performing an UPDATE operation.

To use a matching index scan to update an index in which its key columns are being updated, the following conditions must be met:

- Each updated key column must have a corresponding predicate of the form ″index_key_column = constant″ or ″index_key_column IS NULL″.
- If a view is involved, WITH CHECK OPTION must not be specified.

With list prefetch or multiple index access, any index or indexes can be used in an UPDATE operation. Of course, to be chosen, those access paths must provide efficient access to the data

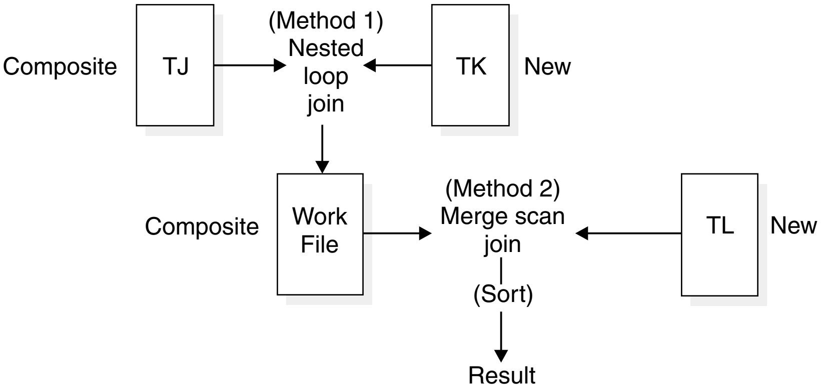
## Interpreting access to two or more tables (join)

A *join* operation retrieves rows from more than one table and combines them. The operation specifies at least two tables, but they need not be distinct.

This section begins with “Definitions and examples” on page 813, below, and continues with descriptions of the methods of joining that can be indicated in a plan table:

- “Nested loop join (METHOD=1)” on page 815
- “Merge scan join (METHOD=2)” on page 816
- “Hybrid join (METHOD=4)” on page 818
- “Star schema (star join)” on page 820

## Definitions and examples



| METHOD | TNAME | ACCESS-TYPE | MATCH-COLS | ACCESS-NAME | INDEX-ONLY | TSLOCK-MODE |
|---|---|---|---|---|---|---|
| 0 | TJ | I | 1 | TJX1 | N | IS |
| 1 | TK | I | 1 | TKX1 | N | IS |
| 2 | TL | I | 0 | TLX1 | Y | S |
| 3 | | | 0 | | N | |

| SORTN UNIQ | SORTN JOIN | SORTN ORDERBY | SORTN GROUPBY | SORTC UNIQ | SORTC JOIN | SORTC ORDERBY | SORTC GROUPBY |
|---|---|---|---|---|---|---|---|
| N | N | N | N | N | N | N | N |
| N | N | N | N | N | N | N | N |
| N | Y | N | N | N | Y | N | N |
| N | N | N | N | N | N | Y | N |

*Figure 103. Join methods as displayed in a plan table*

A join operation can involve more than two tables. But the operation is carried out in a series of steps. Each step joins only two tables.

***Definitions:*** The *composite table* (or *outer table*) in a join operation is the table remaining from the previous step, or it is the first table accessed in the first step. (In the first step, then, the composite table is composed of only one table.) The *new table* (or *inner table*) in a join operation is the table newly accessed in the step.

***Example:*** Figure 103 shows a subset of columns in a plan table. In four steps, DB2:

1. Accesses the first table (METHOD=0), named TJ (TNAME), which becomes the composite table in step 2.
2. Joins the new table TK to TJ, forming a new composite table.
3. Sorts the new table TL (SORTN_JOIN=Y) and the composite table (SORTC_JOIN=Y), and then joins the two sorted tables.
4. Sorts the final composite table (TNAME is blank) into the desired order (SORTC_ORDERBY=Y).

***Definitions:*** A join operation typically matches a row of one table with a row of another on the basis of a *join condition*. For example, the condition might specify that the value in column A of one table equals the value of column X in the other table (`WHERE T1.A = T2.X`).

Two kinds of joins differ in what they do with rows in one table that do not match on the join condition with any row in the other table:

- An *inner join* discards rows of either table that do not match any row of the other table.
- An *outer join* keeps unmatched rows of one or the other table, or of both. A row in the composite table that results from an unmatched row is filled out with null values. Outer joins are distinguished by which unmatched rows they keep.

*Table 113. Join types and kept unmatched rows*

| This outer join: | Keeps unmatched rows from: |
|---|---|
| Left outer join | The composite (outer) table |
| Right outer join | The new (inner) table |
| Full outer join | Both tables |

***Example:*** Figure 104 shows an outer join with a subset of the values it produces in a plan table for the applicable rows. Column JOIN_TYPE identifies the type of outer join with one of these values:

- F for FULL OUTER JOIN
- L for LEFT OUTER JOIN
- Blank for INNER JOIN or no join

At execution, DB2 converts every right outer join to a left outer join; thus JOIN_TYPE never identifies a right outer join specifically.

```
EXPLAIN PLAN SET QUERYNO = 10 FOR
SELECT PROJECT, COALESCE(PROJECTS.PROD#, PRODNUM) AS PRODNUM,
       PRODUCT, PART, UNITS
   FROM PROJECTS LEFT JOIN
      (SELECT PART,
          COALESCE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM,
          PRODUCTS.PRODUCT
          FROM PARTS FULL OUTER JOIN PRODUCTS
             ON PARTS.PROD# = PRODUCTS.PROD#) AS TEMP
   ON PROJECTS.PROD# = PRODNUM
```

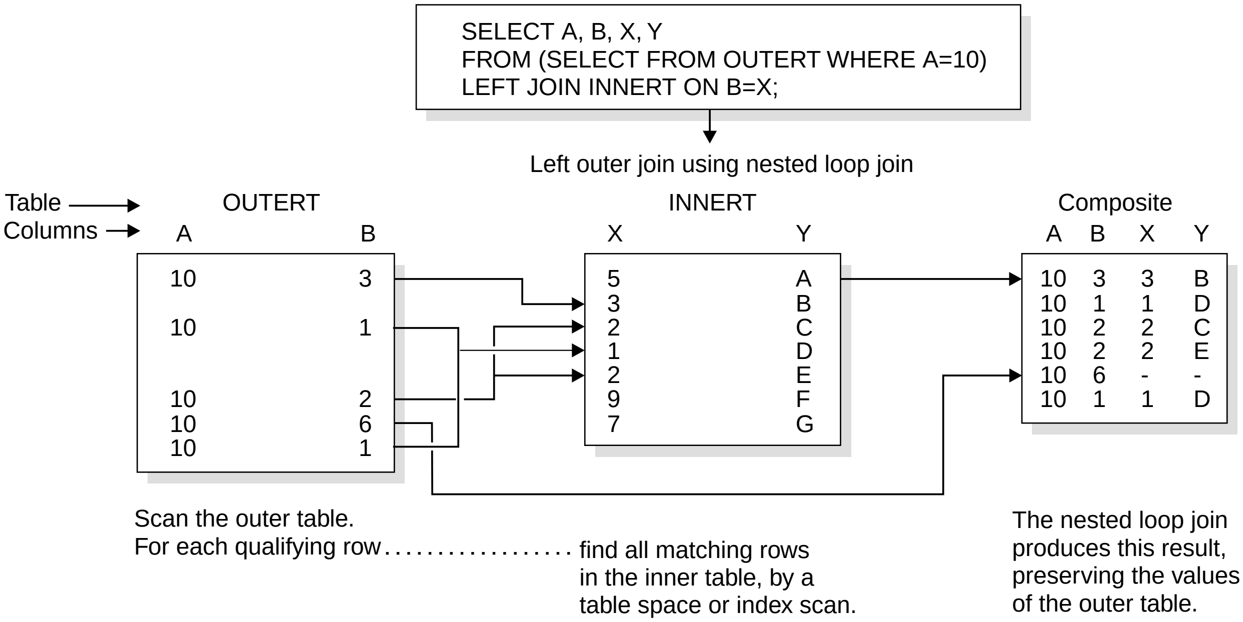
| QUERYNO | QBLOCKNO | PLANNO | TNAME | JOIN_TYPE |
|---|---|---|---|---|
| 10 | 1 | 1 | PROJECTS | |
| 10 | 1 | 2 | TEMP | L |
| 10 | 2 | 1 | PRODUCTS | |
| 10 | 2 | 2 | PARTS | F |

*Figure 104. Plan table output for an example with outer joins*

***Materialization with outer join:*** Sometimes DB2 has to materialize a result table when an outer join is used in conjunction with other joins, views, or nested table expressions. You can tell when this happens by looking at the TABLE_TYPE and TNAME columns of the plan table. When materialization occurs, TABLE_TYPE contains a W, and TNAME shows the name of the materialized table as DSNWFQB(*xx*), where *xx* is the number of the query block (QBLOCKNO) that produced the work file.

## Nested loop join (METHOD=1)

This section describes this common join method.



```
SELECT A, B, X, Y
FROM (SELECT FROM OUTERT WHERE A=10)
LEFT JOIN INNERT ON B=X;
```

Left outer join using nested loop join
Table
Columns
OUTERT
INNERT
Composite

| A | B |
|---|---|
| 10 | 3 |
| 10 | 1 |
| 10 | 2 |
| 10 | 6 |
| 10 | 1 |

| X | Y |
|---|---|
| 5 | A |
| 3 | B |
| 2 | C |
| 1 | D |
| 2 | E |
| 9 | F |
| 7 | G |

| A | B | X | Y |
|---|---|---|---|
| 10 | 3 | 3 | B |
| 10 | 1 | 1 | D |
| 10 | 2 | 2 | C |
| 10 | 2 | 2 | E |
| 10 | 6 | - | - |
| 10 | 1 | 1 | D |

Scan the outer table.
For each qualifying row.................. find all matching rows in the inner table, by a table space or index scan.
The nested loop join produces this result, preserving the values of the outer table.

*Figure 105. Nested Loop Join for a Left Outer Join*

### Method of joining

DB2 scans the composite (outer) table. For each row in that table that qualifies (by satisfying the predicates on that table), DB2 searches for matching rows of the new (inner) table. It concatenates any it finds with the current row of the composite table. If no rows match the current row, then:

For an inner join, DB2 discards the current row.
For an outer join, DB2 concatenates a row of null values.

Stage 1 and stage 2 predicates eliminate unqualified rows during the join. (For an explanation of those types of predicate, see “Stage 1 and stage 2 predicates” on page 716.) DB2 can scan either table using any of the available access methods, including table space scan.

### Performance considerations

The nested loop join repetitively scans the inner table. That is, DB2 scans the outer table once, and scans the inner table as many times as the number of qualifying rows in the outer table. Hence, the nested loop join is usually the most efficient join method when the values of the join column passed to the inner table are in sequence and the index on the join column of the inner table is clustered, or the number of rows retrieved in the inner table through the index is small.

### When it is used

Nested loop join is often used if:

- The outer table is small.
- Predicates with small filter factors reduce the number of qualifying rows in the outer table.

- An efficient, highly clustered index exists on the join columns of the inner table.
- The number of data pages accessed in the inner table is small.

***Example: left outer join:*** Figure 105 on page 815 illustrates a nested loop for a left outer join. The outer join preserves the unmatched row in OUTERT with values A=10 and B=6. The same join method for an inner join differs only in discarding that row.

***Example: one-row table priority:*** For a case like the example below, with a unique index on T1.C2, DB2 detects that T1 has only one row that satisfies the search condition. DB2 makes T1 the first table in a nested loop join.

```
SELECT * FROM T1, T2
   WHERE  T1.C1 = T2.C1 AND
          T1.C2 = 5;
```

***Example: Cartesian join with small tables first:*** A *Cartesian join* is a form of nested loop join in which there are no join predicates between the two tables. DB2 usually avoids a Cartesian join, but sometimes it is the most efficient method, as in the example below. The query uses three tables: T1 has 2 rows, T2 has 3 rows, and T3 has 10 million rows.

```
SELECT * FROM T1, T2, T3
   WHERE  T1.C1 = T3.C1 AND
          T2.C2 = T3.C2 AND
          T3.C3 = 5;
```

Join predicates are between T1 and T3 and between T2 and T3. There is no join predicate between T1 and T2.

Assume that 5 million rows of T3 have the value C3=5. Processing time is large if T3 is the outer table of the join and tables T1 and T2 are accessed for each of 5 million rows.

But if all rows from T1 and T2 are joined, without a join predicate, the 5 million rows are accessed only six times, once for each row in the Cartesian join of T1 and T2. It is difficult to say which access path is the most efficient. DB2 evaluates the different options and could decide to access the tables in the sequence T1, T2, T3.

***Sorting the composite table:*** Your plan table could show a nested loop join that includes a sort on the composite table. DB2 might sort the composite table (the outer table in Figure 105) if the following conditions exist:

- The join columns in the composite table and the new table are not in the same sequence.
- The join column of the composite table has no index.
- The index is poorly clustered.

Nested loop join with a sorted composite table uses sequential detection efficiently to prefetch data pages of the new table, reducing the number of synchronous I/O operations and the elapsed time.

## Merge scan join (METHOD=2)

*Merge scan join* is also known as *merge join* or *sort merge join*. For this method, there must be one or more predicates of the form TABLE1.COL1=TABLE2.COL2, where the two columns have the same data type and length attribute.

## Method of joining

Figure 106 illustrates a merge scan join.

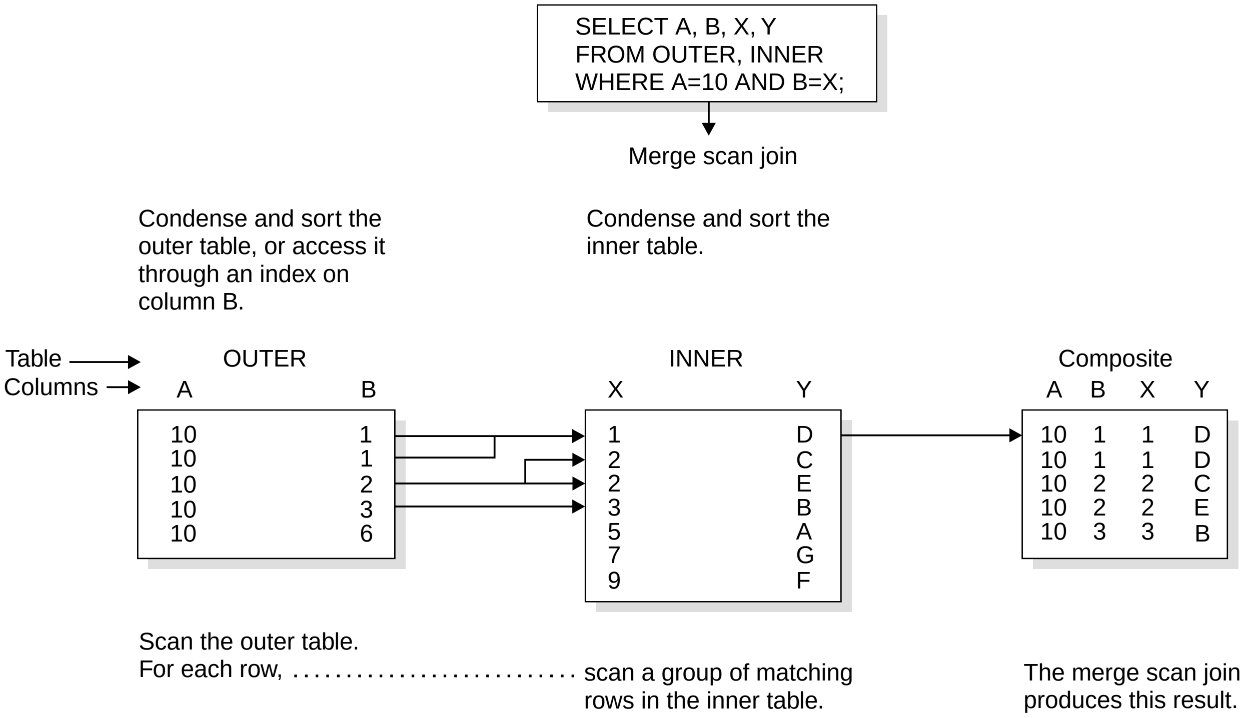


*Figure 106. Merge scan join*

DB2 scans both tables in the order of the join columns. If no efficient indexes on the join columns provide the order, DB2 might sort the outer table, the inner table, or both. The inner table is put into a work file; the outer table is put into a work file only if it must be sorted. When a row of the outer table matches a row of the inner table, DB2 returns the combined rows.

DB2 then reads another row of the inner table that might match the same row of the outer table and continues reading rows of the inner table as long as there is a match. When there is no longer a match, DB2 reads another row of the outer table.

- If that row has the same value in the join column, DB2 reads again the matching group of records from the inner table. Thus, a group of duplicate records in the inner table is scanned as many times as there are matching records in the outer table.
- If the outer row has a new value in the join column, DB2 searches ahead in the inner table. It can find any of the following rows:
  - Unmatched rows in the inner table, with lower values in the join column.
  - A new matching inner row. DB2 then starts the process again.
  - An inner row with a higher value of the join column. Now the row of the outer table is unmatched. DB2 searches ahead in the outer table, and can find any of the following rows:
    - Unmatched rows in the outer table.
    - A new matching outer row. DB2 then starts the process again.
    - An outer row with a higher value of the join column. Now the row of the inner table is unmatched, and DB2 resumes searching the inner table.

If DB2 finds an unmatched row:

For an inner join, DB2 discards the row.

For a left outer join, DB2 discards the row if it comes from the inner table and keeps it if it comes from the outer table.

For a full outer join, DB2 keeps the row.

When DB2 keeps an unmatched row from a table, it concatenates a set of null values as if that matched from the other table. A merge scan join must be used for a full outer join.

### Performance considerations

A full outer join by this method uses all predicates in the ON clause to match the two tables and reads every row at the time of the join. Inner and left outer joins use only stage 1 predicates in the ON clause to match the tables. If your tables match on more than one column, it is generally more efficient to put all the predicates for the matches in the ON clause, rather than to leave some of them in the WHERE clause.

For an inner join, DB2 can derive extra predicates for the inner table at bind time and apply them to the sorted outer table to be used at run time. The predicates can reduce the size of the work file needed for the inner table.

If DB2 has used an efficient index on the join columns, to retrieve the rows of the inner table, those rows are already in sequence. DB2 puts the data directly into the work file without sorting the inner table, which reduces the elapsed time.

### When it is used

A merge scan join is often used if:

- The qualifying rows of the inner and outer table are large, and the join predicate does not provide much filtering; that is, in a many-to-many join.
- The tables are large and have no indexes with matching columns.
- Few columns are selected on inner tables. This is the case when a DB2 sort is used. The fewer the columns to be sorted, the more efficient the sort is.

## Hybrid join (METHOD=4)

The method applies only to an inner join and requires an index on the join column of the inner table.

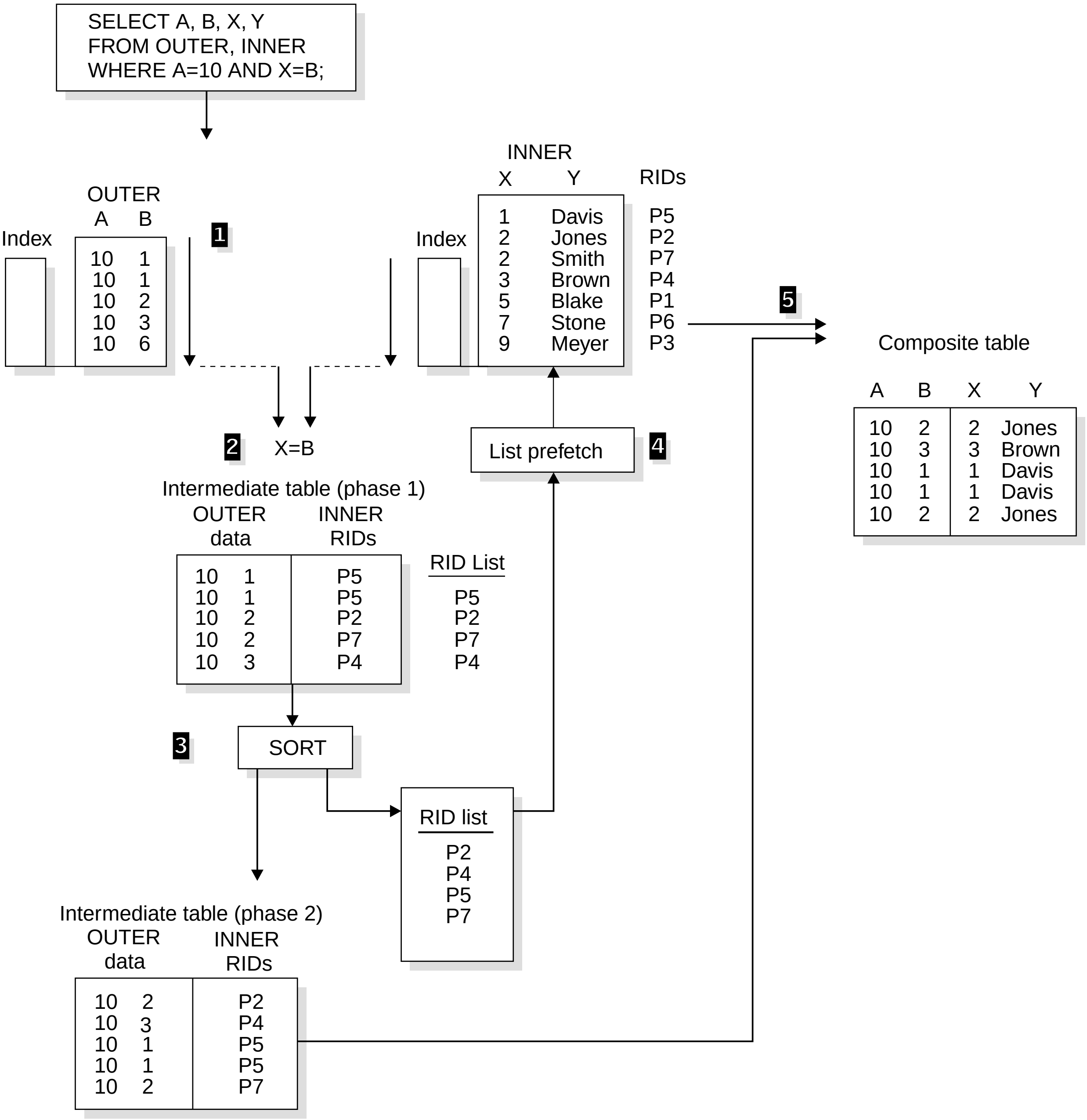


*Figure 107. Hybrid join (SORTN_JOIN='Y')*

## Method of joining

The method requires obtaining RIDs in the order needed to use list prefetch. The steps are shown in Figure 107. In that example, both the outer table (OUTER) and the inner table (INNER) have indexes on the join columns.

In the successive steps, DB2:

1 Scans the outer table (OUTER).

2 Joins the outer tables with RIDs from the index on the inner table. The result is the phase 1 intermediate table. The index of the inner table is scanned for every row of the outer table.

**3** Sorts the data in the outer table and the RIDs, creating a sorted RID list and the phase 2 intermediate table. The sort is indicated by a value of Y in column SORTN_JOIN of the plan table. If the index on the inner table is a clustering index, DB2 can skip this sort; the value in SORTN_JOIN is then N.

**4** Retrieves the data from the inner table, using list prefetch.

**5** Concatenates the data from the inner table and the phase 2 intermediate table to create the final composite table.

### Possible results from EXPLAIN for hybrid join

| Column Value | Explanation |
|---|---|
| **METHOD='4'** | A hybrid join was used. |
| **SORTC_JOIN='Y'** | The composite table was sorted. |
| **SORTN_JOIN='Y'** | The intermediate table was sorted in the order of inner table RIDs. A non-clustered index accessed the inner table RIDs. |
| **SORTN_JOIN='N'** | The intermediate table RIDs were not sorted. A clustered index retrieved the inner table RIDs, and the RIDs were already well ordered. |
| **PREFETCH='L'** | Pages were read using list prefetch. |

### Performance considerations

Hybrid join uses list prefetch more efficiently than nested loop join, especially if there are indexes on the join predicate with low cluster ratios. It also processes duplicates more efficiently because the inner table is scanned only once for each set of duplicate values in the join column of the outer table.

If the index on the inner table is highly clustered, there is no need to sort the intermediate table (SORTN_JOIN=N). The intermediate table is placed in a table in memory rather than in a work file.

### When it is used

Hybrid join is often used if:

- A nonclustered index or indexes are used on the join columns of the inner table.
- The outer table has duplicate qualifying rows.

## Star schema (star join)

A star schema or star join is a logical database design that is included in decision support applications. A star schema is composed of a fact table and a number of dimension tables (or dimension snowflakes) that are connected to it. Normally, a dimension table contains several columns that are given an unique ID column, which is used in the fact table instead of all the values.

You can think of the fact table, which is much larger than the dimension tables, as being in the center surrounded by dimension tables; the result resembles a star formation. The following diagram illustrates the star formation:

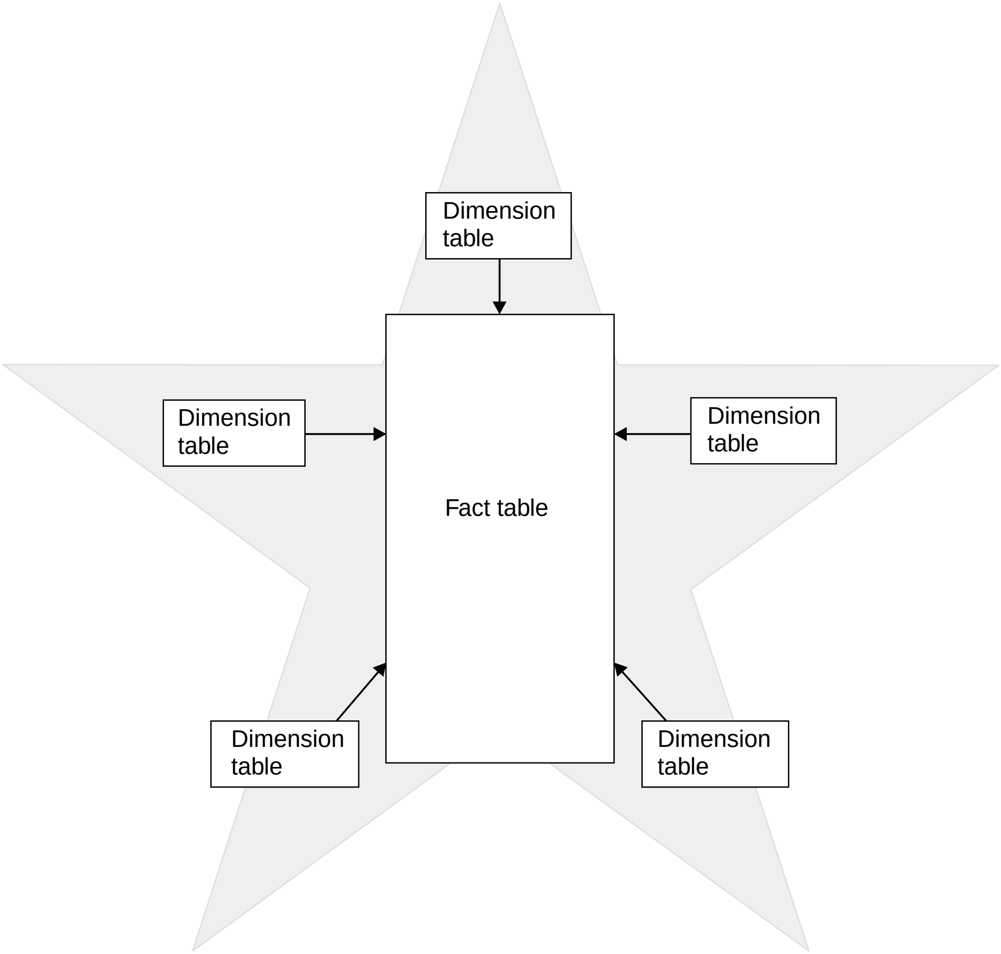


*Figure 108. Star schema with a fact table and dimension tables*

## Example

For an example of a star schema, consider the following scenario. A star schema is composed of a fact table for sales, with dimension tables connected to it for time, products, and geographic locations. The time table has an ID for each month, its quarter, and the year. The product table has an ID for each product item and its class and its inventory. The geographic location table has an ID for each city and its country.

In this scenario, the sales table contains three columns with IDs from the dimension tables for time, product, and location instead of three columns for time, three columns for products, and two columns for location. Thus, the size of the fact table is greatly reduced. In addition, if you needed to change an item, you would do it once in a dimension table instead of several times for each instance of the item in the fact table.

You can create even more complex star schemas by breaking a dimension table into a fact table with its own dimension tables. The fact table would be connected to the main fact table.

### When it is used

To access the data in a star schema, you write SELECT statements that include join operations between the fact table and the dimension tables; no join operations exist between dimension tables. When the query meets the following conditions, that query is a star schema:

- The query references at least two dimensions.
- All join predicates are between the fact table and the dimension tables, or within tables of the same dimension.
- All join predicates between the fact table and dimension tables are equi-join predicates.
- All join predicates between the fact table and dimension tables are Boolean term predicates. For more information, see “Boolean term (BT) predicates” on page 716.
- No correlated subqueries cross dimensions.
- No single fact table column is joined to columns of different dimension tables in join predicates. For example, fact table column `F1` cannot be joined to column `D1` of dimension table T1 and also joined to column `D2` of dimension table T2.
- After DB2 simplifies join operations, no outer join operations exist. For more information, see “When DB2 simplifies join operations” on page 728.
- The data type and length of both sides of a join predicate are the same.
- The value of subsystem parameter STARJOIN is 1, or the cardinality of the fact table to the largest dimension table meets the requirements specified by the value of the subsystem parameter. The values of STARJOIN and cardinality requirements are:

  **-1** Star join is disabled. This is the default.

  **1** Star join is enabled. The one table with the largest cardinality is the fact table. However, if there is more than one table with this cardinality, star join is not enabled.

  **0** Star join is enabled if the cardinality of the fact table is at least 25 times the cardinality of the largest dimension that is a base table that is joined to the fact table.

  *n* Star join is enabled if the cardinality of the fact table is at least *n* times the cardinality of the largest dimension that is a base table that is joined to the fact table, where 2≤*n*≤32768.

- The number of tables in the star schema query block, including the fact table, dimensions tables, and snowflake tables, meet the requirements specified by the value of subsystem parameter SJTABLES. The value of SJTABLES is considered only if the subsystem parameter STARJOIN qualifies the query for star join. The values of SJTABLES are:

  **0** Star join is considered if the query block has 10 or more tables. This is the default.

  **1, 2, or 3** Star join is always considered.

  **4 to 255** Star join is considered if the query block has at least the specified number of tables.

  **226 and greater**
  Star join will never be considered.

  Star join, which can reduce bind time significantly, does not provide optimal performance in all cases. Performance of star join depends on a number of factors such as the available indexes on the fact table, the cluster ratio of the indexes, and the selectivity of rows through local and join predicates. Follow these general guidelines for setting the value of SJTABLES:

- – If you have star schema queries with less than 10 tables and you want to make the star join method applicable to all qualified queries, set the value of SJTABLES to a low number, such as 5.
- – If you have some star schema queries that are not necessarily suitable for star join but want to use star join for relatively large queries, use the default. The star join method will be considered for all qualified queries that have 10 or more tables.
- – If you have star schema queries but, in general, do not want to use star join, consider setting SJTABLES to a higher number, such as 15, if you want to drastically cut the bind time for large queries and avoid a potential bind time SQL return code -101 for large qualified queries.

For recommendations on indexes for star schemas, see “Creating indexes for efficient star schemas” on page 752.

***Examples: query with three dimension tables:*** Suppose you have a store in San Jose and want information about sales of audio equipment from that store in 2000. For this example, you want to join the following tables:

- A fact table for SALES (S)
- A dimension table for TIME (T) with columns for an ID, month, quarter, and year
- A dimension table for geographic LOCATION (L) with columns for an ID, city, region, and country
- A dimension table for PRODUCT (P) with columns for an ID, product item, class, and inventory

You could write the following query to join the tables:

```
SELECT *
FROM SALES S, TIME T, PRODUCT P, LOCATION L
WHERE S.TIME = T.ID  AND
S.PRODUCT = P.ID AND
S.LOCATION = L.ID  AND
T.YEAR = 2000 AND
P.CLASS = 'SAN JOSE';
```

You would use the following index:

```
CREATE INDEX  XSALES_TPL ON SALES (TIME, PRODUCT, LOCATION);
```

Your EXPLAIN output looks like the following table;

| QUERYNO | QBLOCKNO | METHOD | TNAME | JOIN TYPE | SORTN JOIN |
|---|---|---|---|---|---|
| 1 | 1 | 0 | TIME | S | |
| 1 | 1 | 1 | PRODUCT | S | Y |
| 1 | 1 | 1 | LOCATION | S | Y |
| 1 | 1 | 1 | SALES | S | |

*Figure 109. Plan table output for a star join example with TIME, PRODUCT, and LOCATION*

For another example, suppose you want to use the same SALES (S), TIME (T), PRODUCT (P), and LOCATION (L) tables for a similar query and index; however, for this example the index does not include the TIME dimension. A query doesn't have to involve all dimensions. In this example, the star join is performed on one query block at stage 1 and a star join is performed on another query block at stage 2.

You could write the following query to join the tables:

```
SELECT *
FROM SALES S, TIME T, PRODUCT P, LOCATION L
WHERE S.TIME = T.ID  AND
S.PRODUCT = P.ID AND
S.LOCATION = L.ID  AND
T.YEAR = 2000 AND
P.CLASS = 'AUDIO';
```

You would use the following index:

```
CREATE INDEX  XSALES_TPL ON SALES (PRODUCT, LOCATION);
```

Your EXPLAIN output looks like the following table;

| QUERYNO | QBLOCKNO | METHOD | TNAME | JOIN TYPE | SORTN JOIN |
|---|---|---|---|---|---|
| 1 | 1 | 0 | TIME | S | |
| 1 | 1 | 2 | DSNWFQB(02) | S (Note 1) | Y |
| 1 | 2 | 0 | PRODUCT | S (Note 2) | |
| 1 | 2 | 1 | LOCATION | S (Note 2) | |
| 1 | 2 | 1 | SALES | S (Note 2) | |

**Notes to Figure 110:**

1. This star join is handled at stage 2; the tables in this query block are joined with a merge scan join (METHOD = 2).
2. This star join is handled at stage 1; the tables in this query block are joined with a nested loop join (METHOD = 1).

*Figure 110. Plan table output for a star join example with PRODUCT and LOCATION*

## Interpreting data prefetch

*Prefetch* is a mechanism for reading a set of pages, usually 32, into the buffer pool with only one asynchronous I/O operation. Prefetch can allow substantial savings in both processor cycles and I/O costs. To achieve those savings, monitor the use of prefetch.

A plan table can indicate the use of two kinds of prefetch:

- "Sequential prefetch (PREFETCH=S)"
- "List prefetch (PREFETCH=L)" on page 825

If DB2 does not choose prefetch at bind time, it can sometimes use it at execution time nevertheless. The method is described in "Sequential detection at execution time" on page 826.

## Sequential prefetch (PREFETCH=S)

*Sequential prefetch* reads a sequential set of pages. The maximum number of pages read by a request issued from your application program is determined by the size of the buffer pool used. For each buffer pool size (4 KB, 8 KB, 16 KB, and 32

KB), Table 114 shows the number pages read by prefetch for each asynchronous I/O.

*Table 114. The number of pages read by prefetch, by buffer pool size*

| Buffer pool size | Number of buffers | Pages read by prefetch (for each asynchronous I/O) |
|---|---|---|
| 4 KB | <=223 buffers | 8 pages |
| | 224-999 buffers | 16 pages |
| | 1000+ buffers | 32 pages |
| 8 KB | <=112 buffers | 4 pages |
| | 113-499 buffers | 8 pages |
| | 500+ buffers | 16 pages |
| 16 KB | <=56 buffers | 2 pages |
| | 57-249 buffers | 4 pages |
| | 250+ buffers | 8 pages |
| 32 KB | <=16 buffers | 0 pages (prefetch disabled) |
| | 17-99 buffers | 2 pages |
| | 100+ buffers | 4 pages |

For certain utilities (LOAD, REORG, RECOVER), the prefetch quantity can be twice as much.

***When it is used:*** Sequential prefetch is generally used for a table space scan.

For an index scan that accesses 8 or more consecutive data pages, DB2 requests sequential prefetch at bind time. The index must have a cluster ratio of 80% or higher. Both data pages and index pages are prefetched.

## List prefetch (PREFETCH=L)

*List prefetch* reads a set of data pages determined by a list of RIDs taken from an index. The data pages need not be contiguous. The maximum number of pages that can be retrieved in a single list prefetch is 32 (64 for utilities).

List prefetch can be used in conjunction with either single or multiple index access.

### The access method

List prefetch uses the following three steps:

1. RID retrieval: A list of RIDs for needed data pages is found by matching index scans of one or more indexes.
2. RID sort: The list of RIDs is sorted in ascending order by page number.
3. Data retrieval: The needed data pages are prefetched in order using the sorted RID list.

List prefetch does not preserve the data ordering given by the index. Because the RIDs are sorted in page number order before accessing the data, the data is not retrieved in order by any column. If the data must be ordered for an ORDER BY clause or any other reason, it requires an additional sort.

In a hybrid join, if the index is highly clustered, the page numbers might not be sorted before accessing the data.

List prefetch can be used with most matching predicates for an index scan. IN-list predicates are the exception; they cannot be the matching predicates when list prefetch is used.

### When it is used

List prefetch is used:

- Usually with a single index that has a cluster ratio lower than 80%
- Sometimes on indexes with a high cluster ratio, if the estimated amount of data to be accessed is too small to make sequential prefetch efficient, but large enough to require more than one regular read
- Always to access data by multiple index access
- Always to access data from the inner table during a hybrid join

### Bind time and execution time thresholds

DB2 does not consider list prefetch if the estimated number of RIDs to be processed would take more than 50% of the RID pool when the query is executed. You can change the size of the RID pool in the field RID POOL SIZE on installation panel DSNTIPC. The maximum size of a RID pool is 1000MB. The maximum size of a single RID list is approximately 16 million RIDs. For information on calculating RID pool size, see “Increasing RID pool size” on page 574.

During execution, DB2 ends list prefetching if more than 25% of the rows in the table (with a minimum of 4075) must be accessed. Record IFCID 0125 in the performance trace, mapped by macro DSNDQW01, indicates whether list prefetch ended.

When list prefetch ends, the query continues processing by a method that depends on the current access path.

- For access through a single index or through the union of RID lists from two indexes, processing continues by a table space scan.
- For index access before forming an intersection of RID lists, processing continues with the next step of multiple index access. If no step remains and no RID list has been accumulated, processing continues by a table space scan.

While forming an intersection of RID lists, if any list has 32 or fewer RIDs, intersection stops and the list of 32 or fewer RIDs is used to access the data.

## Sequential detection at execution time

If DB2 does not choose prefetch at bind time, it can sometimes use it at execution time nevertheless. The method is called *sequential detection*.

### When it is used

DB2 can use sequential detection for both index leaf pages and data pages. It is most commonly used on the inner table of a nested loop join, if the data is accessed sequentially.

If a table is accessed repeatedly using the same statement (for example, DELETE in a do-while loop), the data or index leaf pages of the table can be accessed sequentially. This is common in a batch processing environment. Sequential detection can then be used if access is through:

- SELECT or FETCH statements
- UPDATE and DELETE statements
- INSERT statements when existing data pages are accessed sequentially

DB2 can use sequential detection if it did not choose sequential prefetch at bind time because of an inaccurate estimate of the number of pages to be accessed.

Sequential detection is not used for an SQL statement that is subject to referential constraints.

### How to tell whether it was used

A plan table does not indicate sequential detection, which is not determined until run time. You can determine whether sequential detection was used from record IFCID 0003 in the accounting trace or record IFCID 0006 in the performance trace.

### How to tell if it might be used

The pattern of accessing a page is tracked when the application scans DB2 data through an index. Tracking is done to detect situations where the access pattern that develops is sequential or nearly sequential.

The most recent 8 pages are tracked. A page is considered page-sequential if it is within P/2 advancing pages of the current page, where P is the prefetch quantity. P is usually 32.

If a page is page-sequential, DB2 determines further if data access is sequential or nearly sequential. Data access is declared sequential if more than 4 out of the last 8 pages are page-sequential; this is also true for index-only access. The tracking is continuous, allowing access to slip into and out of data access sequential.

When data access sequential is first declared, which is called *initial data access* sequential, three page ranges are calculated as follows:

- Let A be the page being requested. RUN1 is defined as the page range of length P/2 pages starting at A.
- Let B be page A + P/2. RUN2 is defined as the page range of length P/2 pages starting at B.
- Let C be page B + P/2. RUN3 is defined as the page range of length P pages starting at C.

For example, assume page A is 10, the following figure illustrates the page ranges that DB2 calculates.

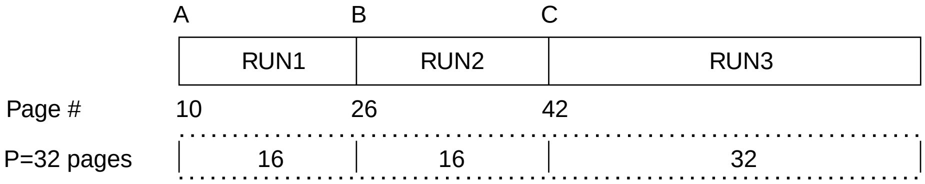


*Figure 111. Initial page ranges to determine when to prefetch*

For initial data access sequential, prefetch is requested starting at page A for P pages (RUN1 and RUN2). The prefetch quantity is always P pages.

For subsequent page requests where the page is 1) page sequential and 2) data access sequential is still in effect, prefetch is requested as follows:

- If the desired page is in RUN1, then no prefetch is triggered because it was already triggered when data access sequential was first declared.

- If the desired page is in RUN2, then prefetch for RUN3 is triggered and RUN2 becomes RUN1, RUN3 becomes RUN2, and RUN3 becomes the page range starting at C+P for a length of P pages.

If a data access pattern develops such that data access sequential is no longer in effect and, thereafter, a new pattern develops that is sequential as described above, then initial data access sequential is declared again and handled accordingly.

Because, at bind time, the number of pages to be accessed can only be estimated, sequential detection acts as a safety net and is employed when the data is being accessed sequentially.

In extreme situations, when certain buffer pool thresholds are reached, sequential prefetch can be disabled. See “Buffer pool thresholds” on page 555 for a description of these thresholds.

# Determining sort activity

DB2 can use two general types of sorts that DB2 can use when accessing data. One is a sort of data rows; the other is a sort of row identifiers (RIDs) in a RID list.

## Sorts of data

After you run EXPLAIN, DB2 sorts are indicated in PLAN_TABLE. The sorts can be either sorts of the composite table or the new table. If a single row of PLAN_TABLE has a ’Y’ in more than one of the sort composite columns, then one sort accomplishes two things. (DB2 will not perform two sorts when two ’Y’s are in the same row.) For instance, if both SORTC_ORDERBY and SORTC_UNIQ are ’Y’ in one row of PLAN_TABLE, then a single sort puts the rows in order and removes any duplicate rows as well.

The only reason DB2 sorts the new table is for join processing, which is indicated by SORTN_JOIN.

### Sorts for group by and order by

These sorts are indicated by SORTC_ORDERBY, and SORTC_GROUPBY in PLAN_TABLE. If there is both a GROUP BY clause and an ORDER BY clause, and if every item in the ORDER-BY list is in the GROUP-BY list, then only one sort is performed, which is marked as SORTC_ORDERBY.

The performance of the sort by the GROUP BY clause is improved when the query accesses a single table and when the GROUP BY column has no index.

### Sorts to remove duplicates

This type of sort is used to process a query with SELECT DISTINCT, with a set function such as COUNT(DISTINCT COL1), or to remove duplicates in UNION processing. It is indicated by SORTC_UNIQ in PLAN_TABLE.

### Sorts used in join processing

Before joining two tables, it is often necessary to first sort either one or both of them. For hybrid join (METHOD 4) and nested loop join (METHOD 1), the composite table can be sorted to make the join more efficient. For merge join (METHOD 2), both the composite table and new table need to be sorted unless an index is used for accessing these tables that gives the correct order already. The sorts needed for join processing are indicated by SORTN_JOIN and SORTC_JOIN in the PLAN_TABLE.

### Sorts needed for subquery processing

When a noncorrelated IN or NOT IN subquery is present in the query, the results of the subquery are sorted and put into a work file for later reference by the parent query. The results of the subquery are sorted because this allows the parent query to be more efficient when processing the IN or NOT IN predicate. Duplicates are not needed in the work file, and are removed. Noncorrelated subqueries used with =ANY or =ALL, or NOT=ANY or NOT=ALL also need the same type of sort as IN or NOT IN subqueries. When a sort for a noncorrelated subquery is performed, you see both SORTC_ORDERBY and SORTC_UNIQUE in PLAN_TABLE. This is because DB2 removes the duplicates and performs the sort.

SORTN_GROUPBY, SORTN_ORDERBY, and SORTN_UNIQ are not currently used by DB2.

## Sorts of RIDs

To perform list prefetch, DB2 sorts RIDs into ascending page number order. This sort is very fast and is done totally in memory. A RID sort is usually not indicated in the PLAN_TABLE, but a RID sort normally is performed whenever list prefetch is used. The only exception to this rule is when a hybrid join is performed and a single, highly clustered index is used on the inner table. In this case SORTN_JOIN is 'N', indicating that the RID list for the inner table was not sorted.

## The effect of sorts on OPEN CURSOR

The type of sort processing required by the cursor affects the amount of time it can take for DB2 to process the OPEN CURSOR statement. This section outlines the effect of sorts and parallelism on OPEN CURSOR.

***Without parallelism:***

- If no sorts are required, then OPEN CURSOR does not access any data. It is at the first fetch that data is returned.
- If a sort is required, then the OPEN CURSOR causes the materialized result table to be produced. Control returns to the application after the result table is materialized. If a cursor that requires a sort is closed and reopened, the sort is performed again.
- If there is a RID sort, but no data sort, then it is not until the first row is fetched that the RID list is built from the index and the first data record is returned. Subsequent fetches access the RID pool to access the next data record.

***With parallelism:***

- At OPEN CURSOR, parallelism is asynchronously started, regardless of whether a sort is required. Control returns to the application immediately after the parallelism work is started.
- If there is a RID sort, but no data sort, then parallelism is not started until the first fetch. This works the same way as with no parallelism.

# Processing for views and nested table expressions

This section describes how DB2 processes views and nested table expressions. A nested table expression (which is called *table expression* in this description) is the specification of a subquery in the FROM clause of an SQL SELECT statement. The processing of table expressions is similar to a view. Two methods are used to satisfy your queries that reference views or table expressions:

- *Merge*
- *Materialization*

You can determine the methods that are used by executing EXPLAIN for the statement that contains the view or nested table expression. In addition, you can use EXPLAIN to determine when UNION operators are used and how DB2 might eliminate unnecessary subselects to improve the performance of a query.

## Merge

The merge process is more efficient than materialization, as described in “Performance of merge versus materialization” on page 835. In the merge process, the statement that references the view or table expression is combined with the fullselect that defined the view or table expression. This combination creates a logically equivalent statement. This equivalent statement is executed against the database.

Consider the following statements, one of which defines a view, the other of which references the view:

```
View-defining statement:               View referencing statement:

CREATE VIEW VIEW1 (VC1,VC21,VC32) AS     SELECT VC1,VC21
SELECT C1,C2,C3 FROM T1                  FROM VIEW1
WHERE C1 > C3;                           WHERE VC1 IN (A,B,C);
```

The fullselect of the view-defining statement can be merged with the view-referencing statement to yield the following logically equivalent statement:

```
Merged statement:

SELECT C1,C2 FROM T1
WHERE C1 > C3 AND C1 IN (A,B,C);
```

Here is another example of when a view and table expression can be merged:

```
SELECT * FROM V1 X
  LEFT JOIN
   (SELECT * FROM T2) Y ON X.C1=Y.C1
                LEFT JOIN T3 Z ON X.C1=Z.C1;
```

```
Merged statement:

SELECT * FROM V1 X
  LEFT JOIN
    T2 ON X.C1 = T2.C1
       LEFT JOIN T3 Z ON X.C1 = Z.C1;
```

## Materialization

Views and table expressions cannot always be merged. Look at the following statements:

```
View defining statement:               View referencing statement:

CREATE VIEW VIEW1 (VC1,VC2) AS           SELECT MAX(VC1)
SELECT SUM(C1),C2 FROM T1                FROM VIEW1;
GROUP BY C2;
```

Column VC1 occurs as the argument of a column function in the view referencing statement. The values of VC1, as defined by the view-defining fullselect, are the result of applying the column function SUM(C1) to groups after grouping the base table T1 by column C2. No equivalent single SQL SELECT statement can be executed against the base table T1 to achieve the intended result. There is no way to specify that column functions should be applied successively.

## Two steps of materialization

In the previous example, DB2 performs materialization of the view or table expression, which is a two step process.

1. The fullselect that defines the view or table expression is executed against the database, and the results are placed in a temporary copy of a result table.
2. The statement that references the view or table expression is then executed against the temporary copy of the result table to obtain the intended result.

Whether materialization is needed depends upon the attributes of the referencing statement, or logically equivalent referencing statement from a prior merge, and the attributes of the fullselect that defines the view or table expression.

## When views or table expressions are materialized

In general, DB2 uses materialization to satisfy a reference to a view or table expression when there is aggregate processing (grouping, column functions, distinct), indicated by the defining fullselect, in conjunction with either aggregate processing indicated by the statement referencing the view or table expression, or by the view or table expression participating in a join. For views and table expressions that are defined with UNION ALL, DB2 can often distribute aggregate processing, joins, and qualified predicates to avoid materialization. For more information, see “Using EXPLAIN to determine UNION activity and query rewrite” on page 834.

Table 115 indicates some cases in which materialization occurs. DB2 can also use materialization in statements that contain multiple outer joins, outer joins that combine with inner joins, or merges that cause a join of greater than 15 tables.

*Table 115. Cases when DB2 performs view or table expression materialization. The ″X″ indicates a case of materialization. Notes follow the table.*

| **A SELECT FROM a view or a table expression uses...(1)** | **View definition or table expression uses...(2)** | | | | | |
|---|---|---|---|---|---|---|
| | GROUP BY | DISTINCT | Column function | Column function DISTINCT | UNION | UNION ALL(4) |
| Joins (3) | X | X | X | X | X | - |
| GROUP BY | X | X | X | X | X | - |
| DISTINCT | - | X | - | X | X | - |
| Column function (without GROUP BY) | X | X | X | X | X | X |
| Column function DISTINCT | X | X | X | X | X | - |
| SELECT subset of view or table expression columns | - | X | - | - | X | - |

**Notes to Table 115:**

1. If the view is referenced as the target of an INSERT, UPDATE, or DELETE, then view merge is used to satisfy the view reference. Only updatable views can be the target in these statements. See Chapter 5 of *DB2 SQL Reference* for information on which views are read-only (not updatable).

   An SQL statement can reference a particular view multiple times where some of the references can be merged and some must be materialized.
2. If a SELECT list contains a host variable in a table expression, then materialization occurs. For example:

```
SELECT C1 FROM
   (SELECT :HV1 AS C1 FROM T1) X;
```

If a view or nested table expression is defined to contain a user-defined function, and if that user-defined function is defined as NOT DETERMINISTIC or EXTERNAL ACTION, then the view or nested table expression is always materialized.

3. Additional details about materialization with outer joins:
   - If a WHERE clause exists in a view or table expression, and it does not contain a column, materialization occurs. For example:

     ```
     SELECT X.C1 FROM
        (SELECT C1 FROM T1
          WHERE 1=1) X LEFT JOIN T2 Y
                    ON X.C1=Y.C1;
     ```
   - If the outer join is a full outer join and the SELECT list of the view or nested table expression does not contain a standalone column for the column that is used in the outer join ON clause, then materialization occurs. For example:

     ```
     SELECT X.C1 FROM
        (SELECT C1+10 AS C2 FROM T1) X FULL JOIN T2 Y
                    ON X.C2=Y.C2;
     ```
   - If there is no column in a SELECT list of a view or nested table expression, materialization occurs. For example:

     ```
     SELECT X.C1 FROM
        (SELECT 1+2+:HV1. AS C1 FROM T1) X LEFT JOIN T2 Y
                    ON X.C1=Y.C1;
     ```
4. DB2 cannot avoid materialization for UNION ALL in all cases. Some of the situations in which materialization occurs includes:
   - When the view is the operand in an outer join for which nulls are used for non-matching values. This situation happens when the view is either operand in a full outer join, the right operand in a left outer join, or the left operand in a right outer join.
   - If the number of tables would exceed 255 after distribution, then distribution will not occur, and the result will be materialized.

## Using EXPLAIN to determine when materialization occurs

For each reference to a view or table expression that is materialized, rows describing the access path for both steps of the materialization process appear in the PLAN_TABLE. These rows describe the access path used to formulate the temporary result indicated by the view's defining fullselect, and they describe the access to the temporary result as indicated by the referencing statement. The defining fullselect can also refer to views or table expressions that need to be materialized.

When DB2 choses materialization, TNAME contains the name of the view or table expression and TABLE_TYPE contains a W. A value of Q in TABLE_TYPE for the name of a view or nested table expresssion indicates that the materialization was virtual and not actual. (Materialization can be virtual when the view or nested table expression definition contains a UNION ALL that is not distributed.) When DB2 chooses merge, EXPLAIN data for the merged statement appears in PLAN_TABLE; only the names of the base tables on which the view or table expression is defined appear.

***Examples:*** Consider the following statements, which define a view and reference the view. Figure 112 on page 833 shows a subset of columns in a plan table for the query. Notice how TNAME contains the name of the view and TABLE_TYPE contains W to indicate that DB2 chooses materialization for the reference to the view because of the use of SELECT DISTINCT in the view defitinion.

```
View defining statement:

CREATE VIEW V1DIS (SALARY, WORKDEPT) as
  (SELECT DISTINCT SALARY, WORKDEPT FROM DSN8810.EMP)

View referencing statement:

SELECT * FROM DSN8810.DEPT
   WHERE DEPTNO IN (SELECT WORKDEPT FROM V1DIS)
```

| QBLOCKNO | PLANNO | QBLOCK_ TYPE | TNAME | TABLE_ TYPE | METHOD |
|---|---|---|---|---|---|
| 1 | 1 | SELECT | DEPT | T | 0 |
| 2 | 1 | NOCOSUB | V1DIS | W | 0 |
| 2 | 2 | NOCOSUB |  | ? | 3 |
| 3 | 1 | NOCOSUB | EMP | T | 0 |
| 3 | 2 | NOCOSUB |  | ? | 3 |

*Figure 112. Plan table output for an example with view materialization*

As the following statements and sample plan table output show, had the VIEW been defined without DISTINCT, DB2 would choose merge instead of materialization. In the sample output, the name of the view does not appear in the plan table, but the table name on which the view is based does appear.

```
View defining statement:

CREATE VIEW V1NODIS (SALARY, WORKDEPT) as
  (SELECT SALARY, WORKDEPT FROM DSN8810.EMP)

View referencing statement:

SELECT * FROM DSN8810.DEPT
   WHERE DEPTNO IN (SELECT WORKDEPT FROM V1NODIS)
```

| QBLOCKNO | PLANNO | QBLOCK_ TYPE | TNAME | TABLE_ TYPE | METHOD |
|---|---|---|---|---|---|
| 1 | 1 | SELECT | DEPT | T | 0 |
| 2 | 1 | NOCOSUB | EMP | T | 0 |
| 2 | 2 | NOCOSUB |  | ? | 3 |

*Figure 113. Plan table output for an example with view merge*

For an example of when a view definition contains a UNION ALL and DB2 can distribute joins and aggregations and avoid materialization, see “Using EXPLAIN to determine UNION activity and query rewrite” on page 834. When DB2 avoids materialization in such cases, TABLE_TYPE contains a Q to indicate that DB2 uses an intermediate result that is not materialized and TNAME shows the name of this intermediate result as DSNWFQB(*xx*), where *xx* is tthe number of the query block that produced the result.

## Using EXPLAIN to determine UNION activity and query rewrite

For each reference to a view or table expression that is defined with UNION or UNION ALL operators, DB2 tries to rewrite the query into a logically equivalent statement with improved performance by:

- Distributing qualified predicates, joins, and aggregations across the subselects of UNION ALL. Such distribution helps to avoid materialization. No distribution is performed for UNION.
- Eliminating unnecessary subselects of the view or table expression. For DB2 to eliminate subselects, the referencing query and the view or table definition must have predicates that are based on common columns.

The QBLOCK_TYPE column in the plan table indicates union activity. For a UNION ALL, the column contains ’UNIONA’. For UNION, the column contains ’UNION’. When QBLOCK_TYPE=’UNION’, the METHOD column on the same row is set to 3 and the SORTC_UNIQ column is set to ’Y’ to indicate that a sort is necessary to remove duplicates. As with other views and table expressions, the plan table also shows when DB2 uses materialization instead of merge.

***Example:*** Consider the following statements, which define a view, reference the view, and show how DB2 rewrites the referencing statement. Figure 114 on page 835 shows a subset of columns in a plan table for the query. Notice how DB2 eliminates the second subselect of the view definition from the rewritten query and how the plan table indicates this removal by showing a UNION ALL for only the first and third subselect in the view definition. The Q in the TABLE_TYPE column indicates that DB2 does not materialize the view.

```
View defining statement: View is created on three tables that contain weekly data

CREATE VIEW V1 (CUSTNO, CHARGES, DATE) as
  SELECT CUSTNO, CHARGES, DATE
  FROM WEEK1
  WHERE DATE BETWEEN '01/01/2000' And '01/07/2000'
UNION ALL
 SELECT CUSTNO, CHARGES, DATE
  FROM WEEK2
  WHERE DATE BETWEEN '01/08/2000' And '01/14/2000'
UNION ALL
  SELECT CUSTNO, CHARGES, DATE
  FROM WEEK3
  WHERE DATE BETWEEN '01/15/2000' And '01/21/2000';

View referencing statement: For each customer in California, find the average charges
during the first and third Friday of January 2000

SELECT V1.CUSTNO, AVG(V1.CHARGES)
  FROM CUST, V1
  WHERE CUST.CUSTNO=V1.CUSTNO
    AND CUST.STATE='CA'
    AND DATE IN ('01/07/2000','01/21/2000')
  GROUP BY V1.CUSTNO;

Rewritten statement (assuming that CHARGES is defined as NOT NULL):

SELECT CUSTNO_U, SUM(SUM_U)/SUM(CNT_U)
  FROM
  ( SELECT WEEK1.CUSTNO, SUM(CHARGES), COUNT(CHARGES)
      FROM CUST, WEEK1
      Where CUST.CUSTNO=WEEK1.CUSTNO AND CUST.STATE='CA'
            AND DATE BETWEEN '01/01/2000' And '01/07/2000'
            AND DATE IN ('01/07/2000','01/21/2000')
      GROUP BY WEEK1.CUSTNO
    UNION ALL
```

```
        SELECT WEEK3.CUSNTO, SUM(CHARGES), COUNT(CHARGES)
          FROM CUST,WEEK3
          WHERE CUST.CUSTNO=WEEK3 AND CUST.STATE='CA'
                AND DATE BETWEEN '01/15/2000' And '01/21/2000'
                AND DATE IN ('01/07/2000','01/21/2000')
          GROUP BY WEEK3.CUSTNO
      ) AS X(CUSTNO_U,SUM_U,CNT_U)
    GROUP BY CUSNTO_U;
```

| QBLOCKNO | PLANNO | TNAME | TABLE_TYPE | METHOD | QBLOCK TYPE | PARENT QBLOCKNO |
|---|---|---|---|---|---|---|
| 1 | 1 | DSNWFQB(02) | Q | 0 |  | 0 |
| 1 | 2 |  | ? | 3 |  | 0 |
| 2 | 1 |  | ? | 0 | UNIONA | 1 |
| 3 | 1 | CUST | T | 0 |  | 2 |
| 3 | 2 | WEEK1 | T | 1 |  | 2 |
| 4 | 1 | CUST | T | 0 |  | 2 |
| 4 | 2 | WEEK3 | T | 2 |  | 2 |

*Figure 114. Plan table output for an example with a view with UNION ALLs*

## Performance of merge versus materialization

Merge performs better than materialization. For materialization, DB2 uses a table space scan to access the materialized temporary result. DB2 materializes a view or table expression only if it cannot merge.

As described above, materialization is a two-step process with the first step resulting in the formation of a temporary result. The smaller the temporary result, the more efficient is the second step. To reduce the size of the temporary result, DB2 attempts to evaluate certain predicates from the WHERE clause of the referencing statement at the first step of the process rather than at the second step. Only certain types of predicates qualify. First, the predicate must be a simple Boolean term predicate. Second, it must have one of the forms shown in Table 116.

*Table 116. Predicate candidates for first-step evaluation*

| Predicate | Example |
|---|---|
| COL op literal | V1.C1 > hv1 |
| COL IS (NOT) NULL | V1.C1 IS NOT NULL |
| COL (NOT) BETWEEN literal AND literal | V1.C1 BETWEEN 1 AND 10 |
| COL (NOT) LIKE constant (ESCAPE constant) | V1.C2 LIKE ’p\%%’ ESCAPE ’\’ |

**Note:** Where ″op″ is =, <>, >, <, <=, or >=, and literal is either a host variable, constant, or special register. The literals in the BETWEEN predicate need not be identical.

Implied predicates generated through predicate transitive closure are also considered for first step evaluation.

## Estimating a statement's cost

You can use EXPLAIN to populate a statement table, *owner*.DSN_STATEMNT_TABLE, at the same time as your PLAN_TABLE is being populated. DB2 provides cost estimates, in service units and in milliseconds, for SELECT, INSERT, UPDATE, and DELETE statements, both static and dynamic. The estimates do not take into account several factors, including cost adjustments that are caused by parallel processing, or the use of triggers or user-defined functions.

Use the information provided in the statement table to:

- Help you determine if a statement is not going to perform within range of your service-level agreements and to tune accordingly.

  DB2 puts its cost estimate into one of two *cost categories*: category A or category B. Estimates that go into cost category A are the ones for which DB2 has adequate information to make an estimate. That estimate is not likely to be 100% accurate, but is likely to be more accurate than any estimate that is in cost category B.

  DB2 puts estimates into cost category B when it is forced to use default values for its estimates, such as when no statistics are available, or because host variables are used in a query. See the description of the REASON column in Table 117 on page 837 for more information about how DB2 determines into which cost category an estimate goes.
- Give a system programmer a basis for entering service-unit values by which to govern dynamic statements.

  Information about using predictive governing is in "Predictive governing" on page 589 .

This section describes the following tasks to obtain and use cost estimate information from EXPLAIN:

1. "Creating a statement table"
2. "Populating and maintaining a statement table" on page 838
3. "Retrieving rows from a statement table" on page 838
4. "Understanding the implications of cost categories" on page 839

See Part 6 of *DB2 Application Programming and SQL Guide* for more information about how to change applications to handle the SQLCODES associated with predictive governing.

## Creating a statement table

To collect information about a statement's estimated cost, create a table called DSN_STATEMNT_TABLE to hold the results of EXPLAIN. A copy of the statements that are needed to create the table are in the DB2 sample library, under the member name DSNTESC.

Figure 115 on page 837 shows the format of a statement table.

```
CREATE TABLE DSN_STATEMNT_TABLE
  ( QUERYNO           INTEGER      NOT NULL WITH DEFAULT,
    APPLNAME          CHAR(8)      NOT NULL WITH DEFAULT,
    PROGNAME          CHAR(8)      NOT NULL WITH DEFAULT,
    COLLID            CHAR(18)     NOT NULL WITH DEFAULT,
    GROUP_MEMBER      CHAR(8)      NOT NULL WITH DEFAULT,
    EXPLAIN_TIME      TIMESTAMP    NOT NULL WITH DEFAULT,
    STMT_TYPE         CHAR(6)      NOT NULL WITH DEFAULT,
    COST_CATEGORY     CHAR(1)      NOT NULL WITH DEFAULT,
    PROCMS            INTEGER      NOT NULL WITH DEFAULT,
    PROCSU            INTEGER      NOT NULL WITH DEFAULT,
    REASON            VARCHAR(254) NOT NULL WITH DEFAULT);
```

*Figure 115. Format of DSN_STATEMNT_TABLE*

Table 117 shows the content of each column. The first five columns of the DSN_STATEMNT_TABLE are the same as PLAN_TABLE.

*Table 117. Descriptions of columns in DSN_STATEMNT_TABLE*

| Column Name | Description |
|---|---|
| QUERYNO | A number that identifies the statement being explained. See the description of the QUERYNO column in Table 111 on page 792 for more information. If QUERYNO is not unique, the value of EXPLAIN_TIME is unique. |
| APPLNAME | The name of the application plan for the row, or blank. See the description of the APPLNAME column in Table 111 on page 792 for more information. |
| PROGNAME | The name of the program or package containing the statement being explained, or blank. See the description of the PROGNAME column in Table 111 on page 792 for more information. |
| COLLID | The collection ID for the package, or blank. See the description of the COLLID column in Table 111 on page 792 for more information. |
| GROUP_MEMBER | The member name of the DB2 that executed EXPLAIN, or blank. See the description of the GROUP_MEMBER column in Table 111 on page 792 for more information. |
| EXPLAIN_TIME | The time at which the statement is processed. This time is the same as the BIND_TIME column in PLAN_TABLE. |
| STMT_TYPE | The type of statement being explained. Possible values are:<br>**SELECT** SELECT<br>**INSERT** INSERT<br>**UPDATE** UPDATE<br>**DELETE** DELETE<br>**SELUPD** SELECT with FOR UPDATE OF<br>**DELCUR** DELETE WHERE CURRENT OF CURSOR<br>**UPDCUR** UPDATE WHERE CURRENT OF CURSOR |
| COST_CATEGORY | Indicates if DB2 was forced to use default values when making its estimates. Possible values:<br>**A** Indicates that DB2 had enough information to make a cost estimate without using default values.<br>**B** Indicates that some condition exists for which DB2 was forced to use default values. See the values in REASON to determine why DB2 was unable to put this estimate in cost category A. |

*Table 117. Descriptions of columns in DSN_STATEMNT_TABLE (continued)*

| Column Name | Description |
|---|---|
| PROCMS | The estimated processor cost, in milliseconds, for the SQL statement. The estimate is rounded up to the next integer value. The maximum value for this cost is 2147483647 milliseconds, which is equivalent to approximately 24.8 days. If the estimated value exceeds this maximum, the maximum value is reported. |
| PROCSU | The estimated processor cost, in service units, for the SQL statement. The estimate is rounded up to the next integer value. The maximum value for this cost is 2147483647 service units. If the estimated value exceeds this maximum, the maximum value is reported. |
| REASON | A string that indicates the reasons for putting an estimate into cost category B.<br>**HAVING CLAUSE** A subselect in the SQL statement contains a HAVING clause.<br>**HOST VARIABLES** The statement uses host variables, parameter markers, or special registers.<br>**REFERENTIAL CONSTRAINTS** Referential constraints of the type CASCADE or SET NULL exist on the target table of a DELETE statement.<br>**TABLE CARDINALITY** The cardinality statistics are missing for one or more of the tables that are used in the statement.<br>**TRIGGERS** Triggers are defined on the target table of an INSERT, UPDATE, or DELETE statement.<br>**UDF** The statement uses user-defined functions. |

## Populating and maintaining a statement table

You populate a statement table at the same time as you populate the corresponding plan table. For more information, see “Populating and maintaining a plan table” on page 796.

Just as with the plan table, DB2 just adds rows to the statement table; it does not automatically delete rows. INSERT triggers are not activated unless you insert rows yourself using and SQL INSERT statement.

To clear the table of obsolete rows, use DELETE, just as you would for deleting rows from any table. You can also use DROP TABLE to drop a statement table completely.

## Retrieving rows from a statement table

To retrieve all rows in a statement table, you can use a query like the following statement, which retrieves all rows about the statement that is represented by query number 13:

```
SELECT * FROM JOE.DSN_STATEMNT_TABLE
   WHERE QUERYNO = 13;
```

The QUERYNO, APPLNAME, PROGNAME, COLLID, and EXPLAIN_TIME columns contain the same values as corresponding columns of PLAN_TABLE for a given plan. You can use these columns to join the plan table and statement table:

```
SELECT A.*, PROCMS, COST_CATEGORY
 FROM JOE.PLAN_TABLE A, JOE.DSN_STATEMNT_TABLE B
   WHERE A.APPLNAME = 'APPL1' AND
   A.APPLNAME = B.APPLNAME AND
   A.PROGNAME = B.PROGNAME AND
   A.COLLID   = B.COLLID AND
   A.BIND_TIME = B.EXPLAIN_TIME
ORDER BY A.QUERYNO, A.QBLOCKNO, A.PLANNO, A.MIXOPSEQ;
```

## Understanding the implications of cost categories

Cost categories are DB2’s way of differentiating estimates for which adequate information is available from those for which it is not. You probably wouldn’t want to spend a lot of time tuning a query based on estimates that are returned in cost category B, because the actual cost could be radically different based on such things as what value is in a host variable, or how many levels of nested triggers and user-defined functions exist.

Similarly, if system administrators use these estimates as input into the resource limit specification table for governing (either predictive or reactive), they probably would want to give much greater latitude for statements in cost category B than for those in cost category A.

Because of the uncertainty involved, category B statements are also good candidates for reactive governing.

***What goes into cost category B?*** DB2 puts a statement’s estimate into cost category B when any of the following conditions exist:

- The statement has UDFs.
- Triggers are defined for the target table:
  - The statement is INSERT, and insert triggers are defined on the target table.
  - The statement is UPDATE, and update triggers are defined on the target table.
  - The statement is DELETE, and delete triggers are defined on the target table.
- The target table of a delete statement has referential constraints defined on it as the parent table, and the delete rules are either CASCADE or SET NULL.
- The WHERE clause predicate has one of the following forms:
  - COL op literal, and the literal is a host variable, parameter marker, or special register. The operator can be >, >=, <, <=, LIKE, or NOT LIKE.
  - COL BETWEEN literal AND literal where either literal is a host variable, parameter marker, or special register.
  - LIKE with an escape clause that contains a host variable.
- The cardinality statistics are missing for one or more tables that are used in the statement.
- A subselect in the SQL statement contains a HAVING clause.

***What goes into cost category A?*** DB2 puts everything that doesn’t fall into category B into category A.

# Chapter 34. Parallel operations and query performance

When DB2 plans to access data from a table or index in a partitioned table space, it can initiate multiple parallel operations. The response time for data or processor-intensive queries can be significantly reduced.

Query I/O parallelism manages concurrent I/O requests for a single query, fetching pages into the buffer pool in parallel. This processing can significantly improve the performance of I/O-bound queries. I/O parallelism is used only when one of the other parallelism modes cannot be used.

Query CP parallelism enables true multi-tasking within a query. A large query can be broken into multiple smaller queries. These smaller queries run simultaneously on multiple processors accessing data in parallel. This reduces the elapsed time for a query.

To expand even farther the processing capacity available for processor-intensive queries, DB2 can split a large query across different DB2 members in a data sharing group. This is known as Sysplex query parallelism. For more information about Sysplex query parallelism, see Chapter 6 of *DB2 Data Sharing: Planning and Administration*.

DB2 can use parallel operations for processing:
- Static and dynamic queries
- Local and remote data access
- Queries using single table scans and multi-table joins
- Access through an index, by table space scan or by list prefetch
- Sort operations

Parallel operations usually involve at least one table in a partitioned table space. Scans of large partitioned table spaces have the greatest performance improvements where both I/O and central processor (CP) operations can be carried out in parallel.

***Parallelism for partitioned and nonpartitioned table spaces:*** Both partitioned and nonpartitioned table spaces can take advantage of query parallelism. Parallelism is now enabled to include non-clustering indexes. Thus, table access can be run in parallel when the application is bound with ANY and the table is accessed through a non-clustering index.

This chapter contains the following topics:

## Comparing the methods of parallelism

The figures in this section show how the parallel methods compare with sequential prefetch and with each other. All three techniques assume access to a table space with three partitions, P1, P2, and P3. The notations P1, P2, and P3 are partitions of a table space. R1, R2, R3, and so on, are requests for sequential prefetch. The combination P2R1, for example, means the first request from partition 2.

Figure 116 shows **sequential processing**. With sequential processing, DB2 takes the 3 partitions in order, completing partition 1 before starting to process partition 2, and completing 2 before starting 3. Sequential prefetch allows overlap of CP processing with I/O operations, but I/O operations do not overlap with each other. In the example in Figure 116, a prefetch request takes longer than the time to process it. The processor is frequently waiting for I/O.

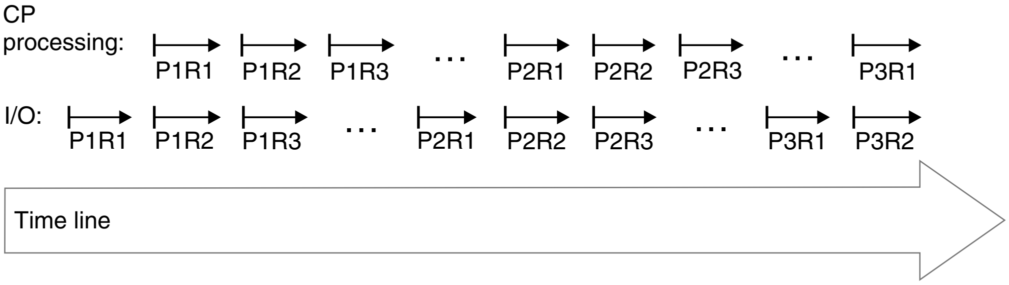


*Figure 116. CP and I/O processing techniques. Sequential processing.*

Figure 117 shows **parallel I/O operations**. With parallel I/O, DB2 prefetches data from the 3 partitions at one time. The processor processes the first request from each partition, then the second request from each partition, and so on. The processor is not waiting for I/O, but there is still only one processing task.

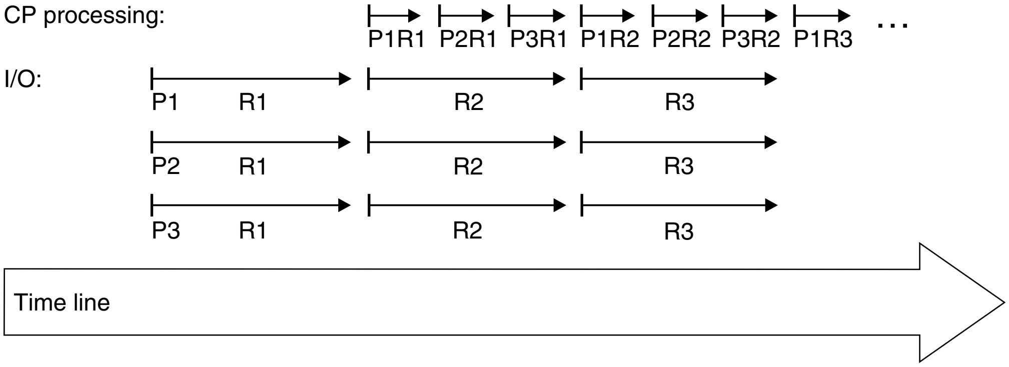


*Figure 117. CP and I/O processing techniques. Parallel I/O processing.*

Figure 118 on page 843 shows **parallel CP processing**. With CP parallelism, DB2 can use multiple parallel tasks to process the query. Three tasks working concurrently can greatly reduce the overall elapsed time for data-intensive and processor-intensive queries. The same principle applies for **Sysplex query parallelism**, except that the work can cross the boundaries of a single CPC.

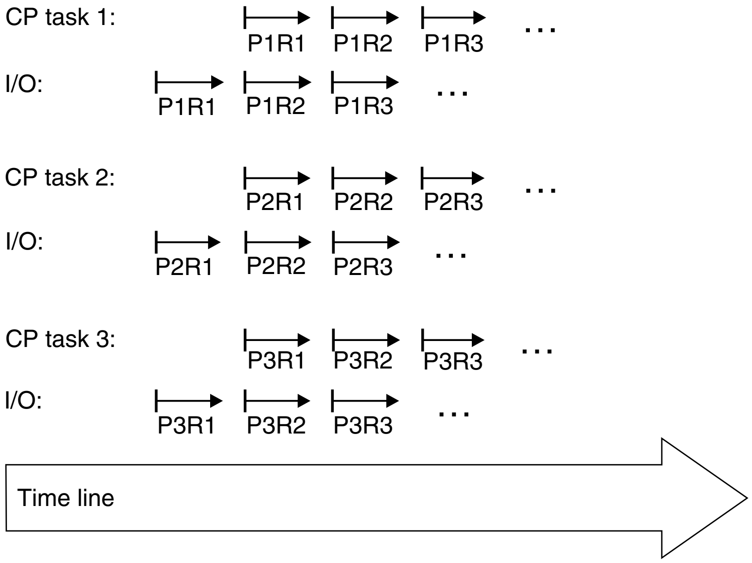


*Figure 118. CP and I/O processing techniques. Query processing using CP parallelism. The tasks can be contained within a single CPC or can be spread out among the members of a data sharing group.*

***Queries that are most likely to take advantage of parallel operations:*** Queries that can take advantage of parallel processing are:

- Those in which DB2 spends most of the time fetching pages—an I/O-intensive query

  A typical I/O-intensive query is something like the following query, assuming that a table space scan is used on many pages:

  ```
  SELECT COUNT(*) FROM ACCOUNTS
   WHERE BALANCE > 0 AND
   DAYS_OVERDUE > 30;
  ```

- Those in which DB2 spends a lot of processor time and also, perhaps, I/O time, to process rows. Those include:

  – *Queries with intensive data scans and high selectivity.* Those queries involve large volumes of data to be scanned but relatively few rows that meet the search criteria.

  – *Queries containing aggregate functions.* Column functions (such as MIN, MAX, SUM, AVG, and COUNT) usually involve large amounts of data to be scanned but return only a single aggregate result.

  – *Queries accessing long data rows.* Those queries access tables with long data rows, and the ratio of rows per page is very low (one row per page, for example).

  – *Queries requiring large amounts of central processor time.* Those queries might be read-only queries that are complex, data-intensive, or that involve a sort.

    A typical processor-intensive query is something like:

    ```
    SELECT MAX(QTY_ON_HAND) AS MAX_ON_HAND,
      AVG(PRICE) AS AVG_PRICE,
      AVG(DISCOUNTED_PRICE) AS DISC_PRICE,
      SUM(TAX) AS SUM_TAX,
      SUM(QTY_SOLD) AS SUM_QTY_SOLD,
      SUM(QTY_ON_HAND - QTY_BROKEN) AS QTY_GOOD,
      AVG(DISCOUNT) AS AVG_DISCOUNT,
      ORDERSTATUS,
      COUNT(*) AS COUNT_ORDERS
    ```

```
FROM   ORDER_TABLE
WHERE SHIPPER = 'OVERNIGHT' AND
      SHIP_DATE < DATE('1996-01-01')
GROUP BY ORDERSTATUS
ORDER BY ORDERSTATUS;
```

***Terminology:*** When the term *task* is used with information on parallel processing, the context should be considered. For parallel query CP processing or Sysplex query parallelism, task is an actual MVS execution unit used to process a query. For parallel I/O processing, a task simply refers to the processing of one of the concurrent I/O streams.

A **parallel group** is the term used to name a particular set of parallel operations (parallel tasks or parallel I/O operations). A query can have more than one parallel group, but each parallel group within the query is identified by its own unique ID number.

The **degree of parallelism** is the number of parallel tasks or I/O operations that DB2 determines can be used for the operations on the parallel group.

In a parallel group, an **originating task** is the TCB (SRB for distributed requests) that coordinates the work of all the **parallel tasks**. Parallel tasks are executable units composed of special SRBs, which are called **preemptable** SRBs.

With preemptable SRBs, the MVS dispatcher can interrupt a task at any time to run other work at the same or higher dispatching priority. For non-distributed parallel work, parallel tasks run under a type of preemptable SRB called a **client** SRB, which lets the parallel task inherit the importance of the originating address space. For distributed requests, the parallel tasks run under a preemptable SRB called an **enclave** SRB. Enclave SRBs are described more fully in “Using Workload Manager to set performance objectives” on page 629.

## Partitioning for optimal parallel performance

This section includes some general considerations for how to partition data for the best performance when using parallel processing. Bear in mind that DB2 does not always choose parallelism, even if you partition the data.

This exercise assumes the following:

- You have narrowed the focus to a few, critical queries that are running sequentially. It is best to include a mix of I/O-intensive and processor-intensive queries into this initial set. You know how long those queries take now and what your performance objectives for those queries are. Although tuning for one set of queries might not work for all queries, overall performance and throughput can be improved.
- You are optimizing for query-at-a-time operations, and you want a query to make use of all the processor and I/O resources available to it.

  When running many queries at the same time, you will probably have to increase the number of partitions and the amount of processing power to achieve similar elapsed times.

This section guides you through the following analyses:

1. Determining the nature of the query (what balance of processing and I/O resources it needs)

2. Determining how many partitions the table space should have to meet your performance objective, number based on the nature of the query and on the processor and I/O configuration at your site

## Determining if a query is I/O- or processor-intensive

To determine if your sequential queries are I/O or processor-intensive, examine the DB2 accounting reports:

- If the “other read I/O time” is close to the total query elapsed time, then the query is I/O-intensive. “Other read I/O time” is the time that DB2 is waiting for pages to be read in to the buffer pools.
- If “CPU time” is close to the total query elapsed time, then the query is processor-intensive.
- If the processor time is somewhere between 30 and 70 percent of the elapsed time, then the query is pretty well-balanced.

## Determining the number of partitions

This section is intended to give you some general guidance. Again, you must take into account the I/O subsystem, the nature of the queries you run, and, if necessary, plan for the data to grow. If your physical and logical design are not closely tied together, thus allowing you to specify any number of partitions, it does no harm to specify more partitions than you need immediately. This approach allows for data and processing resources to grow without you having to repartition the table in the future.

Consider also the operational complexity of managing many partitions. This complexity may not be as much of an issue at sites that use tools, such as the DB2 Automated Utilities Generator and job schedulers.

In general, the number of partitions falls in a range between the number of CPs and the maximum number of I/O paths to the data. When determining the number of partitions that use a mixed set of processor- and I/O-intensive queries, always choose the largest number of partitions in the range you determine.

- **For processor-intensive queries**, specify, at a minimum, a number that is equal to the number of CPs in the system, whether you have a single CPC or multiple CPCs in a data sharing group. If the query is processor-intensive, it can use all CPs available in the system. If you plan to use Sysplex query parallelism, then choose a number that is close to the total number of CPs (including partial allocation of CPs) that you plan to allocate for decision support processing across the data sharing group. Do not include processing resources that are dedicated to other, higher priority, work. For more information about Sysplex query parallelism, see Chapter 6 of *DB2 Data Sharing: Planning and Administration*.
- **For I/O-intensive queries,** calculate the ratio of elapsed time to processor time. Multiply this ratio by the number of processors allocated for decision support processing. Round up this number to determine how many partitions you can use to the best advantage, assuming that these partitions can be on separate devices and have adequate paths to the data. This calculation also assumes that you have adequate processing power to handle the increase in partitions. (This might not be much of an issue with an extremely I/O-intensive query.)

  By partitioning the amount indicated above, the query is brought into balance by reducing the I/O wait time. If the number of partitions is less than the number of CPs available on your system, increase this number close to the number of CPs available. By doing so, other queries that read this same table, but that are more processor-intensive, can take advantage of the additional processing power.

For example, suppose you have a 10-way CPC and the calculated number of partitions is five. Instead of limiting the table space to five partitions, use 10, to equal the number of CPs in the CPC.

***Example configurations for an I/O-intensive query:*** If the I/O cost of your queries is about twice as much as the processing cost, the optimal number of partitions when run on a 10-way processor is 20 (2 * number of processors). Figure 119 shows an I/O configuration that minimizes the elapsed time and allows the CPC to run at 100% busy. It assumes a rule of thumb of four devices per control unit and four channels per control unit.[11]

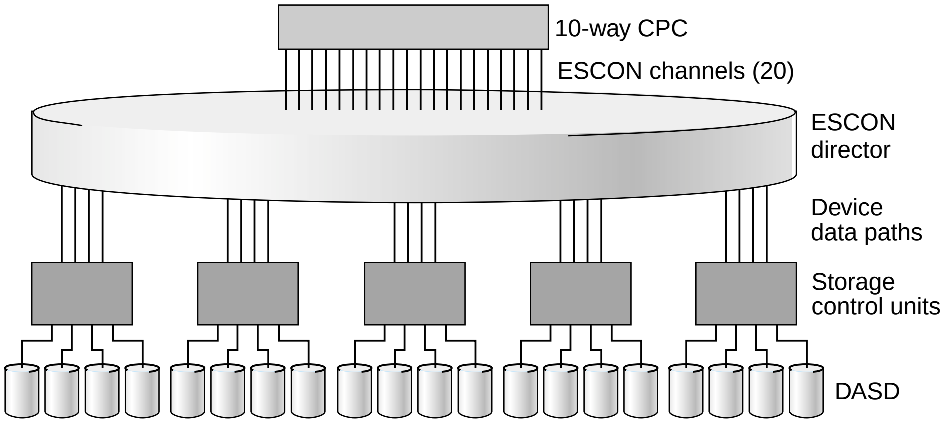


*Figure 119. I/O configuration that maximizes performance for an I/O-intensive query*

## Working with a table space that is already partitioned?

Assume that a table space already has 10 partitions and a particular query uses CP parallelism on a 10-way CPC. When you add “other read I/O wait time” (from accounting class 3) and processing time (from accounting class 2) you determine that I/O cost is three times more than the processing cost. In this case, the optimal number of partitions is 30 (three times more I/O paths). However, if you can run on a data sharing group and you add another DB2 to the group that is running on a 10-way CPC, the I/O configuration that minimizes the elapsed time and allows both CPCs to run at 100% would be 60 partitions.

## Making the partitions the same size

The degree of parallelism is influenced by the size of the largest physical partition. In most cases, DB2 divides the table space into logical pieces, called *work ranges* to differentiate these from physical pieces, based on the size of the largest physical partition of a given table. Suppose that a table consists of 10 000 pages and 10 physical partitions, the largest of which is 5000 pages. DB2 is most likely to create only two work ranges, and the degree of parallelism would be 2. If the same table has evenly sized partitions of 1000 pages each and the query is I/O-intensive, then ten logical work ranges might be created. This example would result in a degree of parallelism of 10 and reduced elapsed time.

[11] A lower-cost configuration could use as few as two to three channels per control unit shared among all controllers using an ESCON® director. However, using four paths minimizes contention and provides the best performance. Paths might also need to be taken offline for service.

DB2 tries to create equal work ranges by dividing the total cost of running the work by the logical partition cost. This division often has some left over work. In this case, DB2 creates an additional task to handle the extra work, rather than making all the work ranges larger, which would reduce the degree of parallelism.

To rebalance partitions that have become skewed, use ALTER INDEX and modify the partitioning range values. This procedure requires a reorganization of the table space.

## Enabling parallel processing

Queries can only take advantage of parallelism if you enable parallel processing. Use the following actions to enable parallel processing:

- For **static SQL**, specify DEGREE(ANY) on BIND or REBIND. This bind option affects static SQL only and does not enable parallelism for dynamic statements.
- For **dynamic SQL**, set the CURRENT DEGREE special register to 'ANY'. Setting the special register affects dynamic statements only. It will have no effect on your static SQL statements. You should also make sure that parallelism is not disabled for your plan, package, or authorization ID in the RLST. You can set the special register with the following SQL statement:

  ```
  SET CURRENT DEGREE='ANY';
  ```

  It is also possible to change the special register default from 1 to ANY for the entire DB2 subsystem by modifying the CURRENT DEGREE field on installation panel DSNTIP4.
- If you bind with isolation CS, choose also the option CURRENTDATA(NO), if possible. This option can improve performance in general, but it also ensures that DB2 will consider parallelism for ambiguous cursors. If you bind with CURRENDATA(YES) and DB2 cannot tell if the cursor is read-only, DB2 does not consider parallelism. It is best to always indicate when a cursor is read-only by indicating FOR FETCH ONLY or FOR READ ONLY on the DECLARE CURSOR statement.
- The virtual buffer pool parallel sequential threshold (VPPSEQT) value must be large enough to provide adequate buffer pool space for parallel processing. For more information on VPPSEQT, see “Buffer pool thresholds” on page 555.

If you enable parallel processing when DB2 estimates a given query’s I/O and central processor cost is high, multiple parallel tasks can be activated if DB2 estimates that elapsed time can be reduced by doing so.

***Special requirements for CP parallelism:*** DB2 must be running on a central processor complex that contains two or more tightly-coupled processors (sometimes called central processors, or CPs). If only one CP is online when the query is bound, DB2 considers only parallel I/O operations.

DB2 also considers only parallel I/O operations if you declare a cursor WITH HOLD and bind with isolation RR or RS. For further restrictions on parallelism, see Table 118 on page 848.

For complex queries, run the query in parallel within a member of a data sharing group. With Sysplex query parallelism, use the power of the data sharing group to process individual complex queries on many members of the data sharing group. For more information on how you can use the power of the data sharing group to run complex queries, see Chapter 6 of *DB2 Data Sharing: Planning and Administration*.

***Limiting the degree of parallelism:*** If you want to limit the maximum number of parallel tasks that DB2 generates, you can use the installation parameter MAX DEGREE in the DSNTIP4 panel. Changing MAX DEGREE, however, is not the way to turn parallelism off. You use the DEGREE bind parameter or CURRENT DEGREE special register to turn parallelism off.

## When parallelism is not used

Parallelism is not used for all queries; for some access paths, it doesn't make sense to incur parallelism overhead. If you are selecting from a temporary table, you won't get parallelism for that, either. If you are not getting parallelism, check Table 118 to see if your query uses any of the access paths that do not allow parallelism.

*Table 118. Checklist of parallel modes and query restrictions*

| | **Is parallelism allowed?** | | | |
|---|---|---|---|---|
| **If query uses this...** | **I/O** | **CP** | **Sysplex** | **Comments** |
| Access via RID list (list prefetch and multiple index access) | Yes | Yes | No | Indicated by an "L" in the PREFETCH column of PLAN_TABLE, or an M, MX, MI, or MQ in the ACCESSTYPE column of PLAN_TABLE. |
| Queries that return LOB values | Yes | Yes | No | |
| Merge scan join on more than one column | No | No | No | |
| Queries that qualify for direct row access | No | No | No | Indicated by D in the PRIMARY_ACCESS_TYPE column of PLAN_TABLE |
| Materialized views or materialized nested table expressions at reference time. | No | No | No | |
| EXISTS within WHERE predicate | No | No | No | |

***DB2 avoids certain hybrid joins when parallelism is enabled:*** To ensure that you can take advantage of parallelism, DB2 does not pick one type of hybrid join (SORTN_JOIN=Y) when the plan or package is bound with CURRENT DEGREE=ANY or if the CURRENT DEGREE special register is set to 'ANY'.

## Interpreting EXPLAIN output

To understand how DB2 plans to use parallelism, examine your PLAN_TABLE output. (Details on all columns in PLAN_TABLE are described in Table 111 on page 792. This section describes a method for examining PLAN_TABLE columns for parallelism and gives several examples.

## A method for examining PLAN_TABLE columns for parallelism

The steps for interpreting the output for parallelism are as follows:

1. **Determine if DB2 plans to use parallelism:**

   For each query block (QBLOCKNO) in a query (QUERYNO), a non-null value in ACCESS_DEGREE or JOIN_DEGREE indicates that some degree of parallelism is planned.

2. **Identify the parallel groups in the query:**

All steps (PLANNO) with the same value for ACCESS_PGROUP_ID, JOIN_PGROUP_ID, SORTN_PGROUP_ID, or SORTC_PGROUP_ID indicate that a set of operations are in the same parallel group. Usually, the set of operations involves various types of join methods and sort operations. Parallel group IDs can appear in the same row of PLAN_TABLE output, or in different rows, depending on the operation being performed. The examples in "PLAN_TABLE examples showing parallelism" help clarify this concept.

3. **Identify the parallelism mode:**

   The column PARALLELISM_MODE tells you the kind of parallelism that is planned (I, C, or X). Within a query block, you cannot have a mixture of "I" and "C" parallel modes. However, a statement that uses more than one query block, such as a UNION, can have "I" for one query block and "C" for another. It is possible to have a mixture of "C" and "X" modes in a query block but not in the same parallel group.

   If the statement was bound while this DB2 is a member of a data sharing group, the PARALLELISM_MODE column can contain "X" even if only this one DB2 member is active. This lets DB2 take advantage of extra processing power that might be available at execution time. If other members are not available at execution time, then DB2 runs the query within the single DB2 member.

## PLAN_TABLE examples showing parallelism

For these examples, the other values would not change whether the PARALLELISM_MODE is I, C, or X.

- **Example 1: single table access**

  Assume that DB2 decides at bind time to initiate three concurrent requests to retrieve data from table T1. Part of PLAN_TABLE appears as follows. If DB2 decides not to use parallel operations for a step, ACCESS_DEGREE and ACCESS_PGROUP_ID contain null values.

| TNAME | METHOD | ACCESS_ DEGREE | ACCESS_ PGROUP_ ID | JOIN_ DEGREE | JOIN_ PGROUP_ ID | SORTC_ PGROUP_ ID | SORTN_ PGROUP_ ID |
|---|---|---|---|---|---|---|---|
| T1 | 0 | 3 | 1 | (null) | (null) | (null) | (null) |

- **Example 2: nested loop join**

  Consider a query that results in a series of nested loop joins for three tables, T1, T2 and T3. T1 is the outermost table, and T3 is the innermost table. DB2 decides at bind time to initiate three concurrent requests to retrieve data from each of the three tables. For the nested loop join method, all the retrievals are in the same parallel group. Part of PLAN_TABLE appears as follows:

| TNAME | METHOD | ACCESS_ DEGREE | ACCESS_ PGROUP_ ID | JOIN_ DEGREE | JOIN_ PGROUP_ ID | SORTC_ PGROUP_ ID | SORTN_ PGROUP_ ID |
|---|---|---|---|---|---|---|---|
| T1 | 0 | 3 | 1 | (null) | (null) | (null) | (null) |
| T2 | 1 | 3 | 1 | 3 | 1 | (null) | (null) |
| T3 | 1 | 3 | 1 | 3 | 1 | (null) | (null) |

- **Example 3: merge scan join**

  Consider a query that causes a merge scan join between two tables, T1 and T2. DB2 decides at bind time to initiate three concurrent requests for T1 and six concurrent requests for T2. The scan and sort of T1 occurs in one parallel group.

The scan and sort of T2 occurs in another parallel group. Furthermore, the merging phase can potentially be done in parallel. Here, a third parallel group is used to initiate three concurrent requests on each intermediate sorted table. Part of PLAN_TABLE appears as follows:

| TNAME | METHOD | ACCESS_ DEGREE | ACCESS_ PGROUP_ ID | JOIN_ DEGREE | JOIN_ PGROUP_ ID | SORTC_ PGROUP_ ID | SORTN_ PGROUP_ ID |
|---|---|---|---|---|---|---|---|
| T1 | 0 | 3 | 1 | (null) | (null) | (null) | (null) |
| T2 | 2 | 6 | 2 | 3 | 3 | 1 | 2 |

- **Example 4: hybrid join**

  Consider a query that results in a hybrid join between two tables, T1 and T2. Furthermore, T1 needs to be sorted; as a result, in PLAN_TABLE the T2 row has SORTC_JOIN=Y. DB2 decides at bind time to initiate three concurrent requests for T1 and six concurrent requests for T2. Parallel operations are used for a join through a clustered index of T2.

  Because T2's RIDs can be retrieved by initiating concurrent requests on the partitioned index, the joining phase is a parallel step. The retrieval of T2's RIDs and T2's rows are in the same parallel group. Part of PLAN_TABLE appears as follows:

| TNAME | METHOD | ACCESS_ DEGREE | ACCESS_ PGROUP_ ID | JOIN_ DEGREE | JOIN_ PGROUP_ ID | SORTC_ PGROUP_ ID | SORTN_ PGROUP_ ID |
|---|---|---|---|---|---|---|---|
| T1 | 0 | 3 | 1 | (null) | (null) | (null) | (null) |
| T2 | 4 | 6 | 2 | 6 | 2 | 1 | (null) |

## Monitoring parallel operations

The number of parallel operations or tasks used to access data is initially determined at bind time, and later adjusted when the query is executed.

***Bind time:*** At bind time, DB2 collects partition statistics from the catalog, estimates the processor cycles for the costs of processing the partitions, and determines the optimal number of parallel tasks to achieve minimum elapsed time.

When a planned degree exceeds the number of online CPs, it can mean that the query is not completely processor-bound and is instead approaching the number of partitions because it is I/O-bound. In general, the more I/O-bound a query is, the closer the degree of parallelism is to the number of partitions.

In general, the more processor-bound a query is, the closer the degree of parallelism is to the number of online CPs, and it can even exceed the number of CPs by one. For example, assume that you have a processor-intensive query on a 10-partition table, and that this query is running on a 6-way CPC. It is possible for the degree of parallelism to be up to 7 in this case.

To help DB2 determine the optimal degree of parallelism, use the utility RUNSTATS to keep your statistics current.

PLAN_TABLE shows the planned degree of parallelism in the columns ACCESS_DEGREE and JOIN_DEGREE.

***Execution time:*** For each parallel group, parallelism (either CP or I/O) can execute at a reduced degree or degrade to sequential operations for the following reasons:

- Amount of virtual buffer pool space available
- Host variable values
- Availability of the hardware sort assist facility
- Ambiguous cursors
- A change in the number or configuration of online processors
- The join technique that DB2 uses (I/O parallelism not supported when DB2 uses the star join technique)

At execution time, it is possible for a plan using Sysplex query parallelism to use CP parallelism. All parallelism modes can degenerate to a sequential plan. No other changes are possible.

## Using DISPLAY BUFFERPOOL

You can use the output from DISPLAY BUFFERPOOL DETAIL report to see how well the buffer pool is able to satisfy parallel operations.

```
DSNB440I = PARALLEL ACTIVITY -
            PARALLEL REQUEST =      282  DEGRADED PARALLEL=       5
```

The PARALLEL REQUEST field in this example shows that DB2 was negotiating buffer pool resource for 282 parallel groups. Of those 282 groups, only 5 were degraded because of a lack of buffer pool resource. A large number in the DEGRADED PARALLEL field could indicate that there are not enough buffers that can be used for parallel processing.

## Using DISPLAY THREAD

DISPLAY THREAD displays parallel tasks. Whereas previously you would only see information about the originating task, now you can see information about the parallel tasks associated with that originating task. The status field contains PT for parallel tasks. All parallel tasks are displayed immediately after their corresponding originating thread.

See Chapter 2 of *DB2 Command Reference* for information about the syntax of the command DISPLAY THREAD.

## Using DB2 trace

The statistics trace indicates when parallel groups do not run to the planned degree or run sequentially. These are possible indicators that there are queries that are not achieving the best possible response times. Use the accounting trace to ensure that your parallel queries are meeting their response time goals. If there appears to be a problem with a parallel query, then use the performance trace to do further analysis.

### Accounting trace

By default, DB2 rolls task accounting into an accounting record for the originating task. DB2 PM also summarizes all accounting records generated for a parallel query and presents them as one logical accounting record. DB2 PM presents the times for the originating tasks separately from the accumulated times for all the parallel tasks.

As shown in Figure 120 on page 852 CPU TIME-AGENT is the time for the originating tasks, while CPU TIME-PAR.TASKS ( **A** ) is the accumulated processing time for the parallel tasks.

```
TIMES/EVENTS  APPL (CLASS 1)   DB2 (CLASS 2)    CLASS 3 SUSP.   ELAPSED TIME
------------  --------------  --------------    --------------  ------------
ELAPSED TIME       32.578741       32.312218    LOCK/LATCH         25.461371
 NON-NESTED        28.820003       30.225885    SYNCHRON. I/O       0.142382
 STORED PROC        3.758738        2.086333     DATABASE I/O       0.116320
 UDF                0.000000        0.000000     LOG WRTE I/O       0.026062
 TRIGGER            0.000000        0.000000    OTHER READ I/O   3:00.404769
                                                OTHER WRTE I/O      0.000000
CPU TIME         1:29.695300     1:29.644026    SER.TASK SWTCH      0.000000
 AGENT              0.225153        0.178128     UPDATE COMMIT      0.000000
  NON-NESTED        0.132351        0.088834     OPEN/CLOSE         0.000000
  STORED PROC       0.092802        0.089294     SYSLGRNG REC       0.000000
  UDF               0.000000        0.000000     EXT/DEL/DEF        0.000000
  TRIGGER           0.000000        0.000000     OTHER SERVICE      0.000000
 PAR.TASKS     A 1:29.470147     1:29.465898    ARC.LOG(QUIES)      0.000000
⋮

...            QUERY PARALLEL.     TOTAL
               ---------------  --------
               MAXIMUM MEMBERS         1
               MAXIMUM DEGREE         10
               GROUPS EXECUTED         1
               RAN AS PLANNED  B       1
               RAN REDUCED     C       0
               ONE DB2 COOR=N          0
               ONE DB2 ISOLAT          0
               SEQ - CURSOR   D        0
               SEQ - NO ESA   E        0
               SEQ - NO BUF   F        0
               SEQ - ENCL.SER.         0
               MEMB SKIPPED(%)         0
               DISABLED BY RLF    G   NO
               REFORM PARAL-CONFIG H   0
               REFORM PARAL-NO BUF     0
```

*Figure 120. Partial accounting trace, query parallelism*

As you can see in the report, the values for CPU TIME and I/O WAIT TIME are larger than the elapsed time. It is possible for processor and suspension time to be larger than elapsed time because these times are accumulated from multiple parallel tasks, while the elapsed time is less than it would be if run sequentially.

If you have baseline accounting data for the same thread run without parallelism, the elapsed times and processor times should not be significantly larger when that query is run in parallel. If it is significantly larger, or if response time is poor, you will need to examine the accounting data for the individual tasks. Use the DB2 PM Record Trace for the IFCID 0003 records of the thread you want to examine. Use the performance trace if you need more information to determine the cause of the response time problem.

## Performance trace

The performance trace can give you information about tasks within a group. To determine the actual number of parallel tasks used, refer to field QW0221AD in IFCID 0221, as mapped by macro DSNDQW03. The 0221 record also gives you information about the key ranges used to partition the data.

IFCID 0222 contains the elapsed time information for each parallel task and each parallel group in each SQL query. DB2 PM presents this information in its SQL Activity trace.

If your queries are running sequentially or at a reduced degree because of a lack of buffer pool resources, the QW0221XC field of IFCID 0221 indicates which buffer pool is constrained.

## Tuning parallel processing

Much of the information in this section applies also to Sysplex query parallelism. See Chapter 6 of *DB2 Data Sharing: Planning and Administration* for more information.

If there are many parallel groups that do not run at the planned degree (see **B** in Figure 120 on page 852), check the following factors:

- Buffer pool availability

  Depending on buffer pool availability, DB2 could reduce the degree of parallelism (see **C** in Figure 120 on page 852) or revert to a sequential plan before executing the parallel group (**F** in the figure).

  To determine which buffer pool is short on storage, see section QW0221C in IFCID 0221. You can use the ALTER BUFFERPOOL command to increase the buffer pool space available for parallel operations by modifying the following parameters:
  – VPSIZE, the size of the virtual buffer pool
  – VPSEQT, the sequential steal threshold
  – VPPSEQT, the parallel sequential threshold
  – VPXPSEQT, the assisting parallel sequential threshold, used only for Sysplex query parallelism.

  If the buffer pool is busy with parallel operations, the sequential prefetch quantity might also be reduced.

  The parallel sequential threshold also has an impact on **work file processing** for parallel queries. DB2 assumes that you have all your work files of the same size (4KB or 32KB) in the same buffer pool and makes run time decisions based on a single buffer pool. A lack of buffer pool resources for the work files can lead to a reduced degree of parallelism or cause the query to run sequentially.

  If increasing the parallel thresholds does not help solve the problem of reduced degree, you can increase the total buffer pool size (VPSIZE). Use information from the statistics trace to determine the amount of buffer space you need. Use the following formula:

  ```
  (QBSTJIS / QBSTPQF) × 32 = buffer increase value
  ```

  QBSTJIS is the total number of requested prefetch I/O streams that were denied because of a storage shortage in the buffer pool. (There is one I/O stream per parallel task.) QBSTPQF is the total number of times that DB2 could not allocate enough buffer pages to allow a parallel group to run to the planned degree.

  As an example, assume QBSTJIS is 100000 and QBSTPQF is 2500:

  ```
  (100000 / 2500) × 32 = 1280
  ```

  Use ALTER BUFFERPOOL to increase the current VPSIZE by 2560 buffers to alleviate the degree degradation problem. Use the DISPLAY BUFFERPOOL command to see the current VPSIZE.
- Physical contention

As much as possible, put data partitions on separate physical devices to minimize contention. Try not to use more partitions than there are internal paths in the controller.

- Run time host variables

  A host variable can determine the qualifying partitions of a table for a given query. In such cases, DB2 defers the determination of the planned degree of parallelism until run time, when the host variable value is known.
- Updatable cursor

  At run time, DB2 might determine that an ambiguous cursor is updatable. This appears in D in the accounting report.
- Proper hardware and software support

  If you do not have the hardware sort facility at run time, and a sort merge join is needed, you see a value in E.
- A change in the configuration of online processors

  If there are fewer online processors at run time than at bind time, DB2 reformulates the parallel degree to take best advantage of the current processing power. This reformulation is indicated by a value in H in the accounting report.

***Locking considerations for repeatable read applications:*** For CP parallelism, locks are obtained independently by each task. Be aware that this can possibly increase the total number of locks taken for applications that:

- Use an isolation level of repeatable read
- Use CP parallelism
- Repeatedly access the table space using a lock mode of IS without issuing COMMITs

As is recommended for all repeatable-read applications, be sure to issue frequent COMMITs to release the lock resources that are held. Repeatable read or read stability isolation cannot be used with Sysplex query parallelism.

## Disabling query parallelism

To disable parallel operations, do any of the following actions:

- For static SQL, rebind to change the option DEGREE(ANY) to DEGREE(1). You can do this by using the DB2I panels, the DSN subcommands, or the DSNH CLIST. The default is DEGREE(1).
- For dynamic SQL, execute the following SQL statement:

  ```
  SET CURRENT DEGREE = '1';
  ```

  The default value for CURRENT DEGREE is 1 unless your installation has changed the default for the CURRENT DEGREE special register.
- Set the parallel sequential threshold (VPPSEQT) to 0.
- Add a row to your resource limit facility’s specification table (RLST) for your plan, package, or authorization ID with the RLFFUNC value set to “3” to disable I/O parallelism, “4” to disable CP parallelism, or “5” to disable Sysplex query parallelism. To disable all types of parallelism, you need a row for all three types (assuming that Sysplex query parallelism is enabled on your system.) In a system with a very high processor utilization rate (that is, greater than 98 percent), I/O parallelism might be a better choice because of the increase in processor overhead with CP parallelism. In this case, you could disable CP parallelism for your dynamic queries by putting a “4” in the resource limit specification table for the plan or package.

If you have a Sysplex, you might want to use a “5” to disable Sysplex query parallelism, depending on how high processor utilization is in the members of the data sharing group.

To determine if parallelism has been disabled by a value in your resource limit specification table (RLST), look for a non-zero value in field QXRLFDPA in IFCID 0002 or 0003 (shown in G in Figure 120 on page 852). The QW0022RP field in IFCID 0022 indicates whether this particular statement was disabled. For more information on how the resource limit facility governs modes of parallelism, see “Descriptions of the RLST columns” on page 584.

# Chapter 35. Tuning and monitoring in a distributed environment

This section describes some ways you can tune your systems and applications that use DRDA or DB2 private protocol for distributed data access.

This chapter contains the following sections:

## Understanding remote access types

DB2 supports two different types of remote access between the requesting relational database management system (DBMS) and the serving relational database management system. The two types of access are *DB2 private protocol access* and *DRDA access*. When three-part named objects are referenced (or aliases for three-part name objects are referenced, DB2 chooses between the two connection types based on the bind option you choose (or the default protocol set at your site).

**Recommendation:** Use DRDA for new applications, and begin migrating existing private protocol applications to DRDA. No enhancements are planned for private protocol.

### Characteristics of DRDA

The application can remotely bind packages and can execute packages of static or dynamic SQL that have previously been bound at that location. Distributed processing using DRDA has the following characteristics:

- The application can access data at any server that supports DRDA, not just a DB2 on an OS/390 or z/OS operating system.
- The application can use remote BIND to bind SQL into packages at the serving relational database management system.
- The application can connect to other relational database management systems in the network and execute packages at those database management systems.

### Characteristics of DB2 private protocol

For distributed work between the two subsystems, DB2 uses communications connections that are specific to DB2. Access using DB2 private protocol has these characteristics:

- It does not support many SQL features, including user-defined functions, LOBs, and stored procedures.
- Only DB2 on OS/390 and z/OS subsystems can communicate using this connection.
- When a static SQL statement is passed to the server, it is dynamically bound and then executed. The statement is dynamically bound only the first time it is executed within a unit of recovery. Subsequent executions of the statement in the unit of recovery do not pay the cost of the dynamic bind. However, if the statement is executed again after a COMMIT or ROLLBACK, another dynamic bind occurs.

- Within a unit of work, updates can be made to any number of DB2 subsystems. An application can also read at several sites within a unit of work.

## Tuning distributed applications

A query sent to a remote system can sometimes take longer to execute than the same query, accessing tables of the same size, on the local DB2 subsystem. The principal reasons for this potential increase in execution time are:

- The time required to send messages across the network
- Overhead processing, including startup and communication subsystem session management

Some aspects of overhead processing, for instance, network processing, are not under DB2 control. (Suggestions for tuning your network are in Part 3 of *DB2 Installation Guide*.)

Monitoring and tuning performance in a distributed environment is a complex task involving knowledge of several products. Some guidelines follow for improving the performance of distributed applications. The guidelines are divided into the following areas:

- “The application and the requesting system”
- “The serving system” on page 865

## The application and the requesting system

Minimizing the number of messages sent between the requester and the server is a primary way to improve performance. Three topics that require more extensive description are:

- “Block fetching result sets” on page 859.
- “Optimizing for very large results sets for DRDA” on page 863.
- “Optimizing for small results sets for DRDA” on page 864.

This section describes bind options and SQL statement options to consider

### BIND options

If appropriate for your applications, consider the following options for bind:

- Use the bind option DEFER(PREPARE), which may reduce the number of messages that must be sent back and forth across the network. For more information on using the DEFER(PREPARE) option, see Part 4 of *DB2 Application Programming and SQL Guide*.
- Bind application plans and packages with ISOLATION(CS) whenever possible, which can reduce contention and message overhead.

### SQL statement options

- Avoid using several SQL statements when one well-tuned SQL statement can retrieve the desired results. Alternatively, put your SQL statements in a stored procedure, issue your SQL statements at the server through the stored procedure, and return the result. Using a stored procedure creates only one send and receive operation (for the CALL statement) instead of a potential send and receive operation for each SQL statement.

  Depending on how many SQL statements are in your application, using stored procedures can significantly lower your processor and elapsed time costs. For more information on how to use stored procedures, see Part 6 of *DB2 Application Programming and SQL Guide*.

- Use the SQL statement RELEASE and the bind option DISCONNECT(EXPLICIT). The SQL statement RELEASE minimizes the network traffic needed to release a remote connection at commit time. For example, if the application has connections to several different servers, specify the RELEASE statement when the application has completed processing for each server. The RELEASE statement does not close cursors, release any resources, or prevent further use of the connection until the COMMIT is issued. It just makes the processing at COMMIT time more efficient.

  The bind option DISCONNECT(EXPLICIT) destroys all remote connections for which RELEASE was specified.
- Commit frequently to avoid holding resources at the server.
- Unless you are using dynamic statement caching at the server, avoid using parameter markers in dynamic SELECT statements at a DB2 for OS/390 and z/OS requester—use literals instead. Using literals enables DB2 for OS/390 and z/OS to send the PREPARE and OPEN in one network message. DB2 can send the PREPARE and OPEN in one message, even with parameter markers, if you bind with DEFER(PREPARE).
- Consider carefully using the clause COMMIT ON RETURN YES of the CREATE PROCEDURE statement to indicate that DB2 should issue an implicit COMMIT on behalf of the stored procedure upon return from the CALL statement. Using the clause can reduce the length of time locks are held and can reduce network traffic. With COMMIT ON RETURN YES, any updates made by the client before calling the stored procedure are committed with the stored procedure changes. See Part 6 of *DB2 Application Programming and SQL Guide* for more information.
- When requesting LOB data, set the CURRENT RULES special register to DB2 instead of to STD before performing a CONNECT. A value of DB2, which is the default, can offer performance advantages. When a DB2 for OS/390 and z/OS server receives an OPEN request for a cursor, the server uses the value in the CURRENT RULES special register to determine whether the application intends to switch between LOB values and LOB locator values when fetching different rows in the cursor. If you specify a value of DB2 for CURRENT RULES, the application indicates that the first FETCH request will specify the format for each LOB column in the answer set and that the format will not change in a subsequent FETCH request. However, if you set the value of CURRENT RULES to STD, the application intends to fetch a LOB column into either a LOB locator host variable or a LOB host variable.

  Although a value of STD for CURRENT RULES gives you more programming flexibility when you retrieve LOB data, you can get better performance if you use a value of DB2. With the STD option, the server will not block the cursor, while with the DB2 option it may block the cursor where it is possible to do so. For more information, see “LOB data and its effect on block fetch” on page 861.

## Block fetching result sets

DB2 has an important capability called *block fetch* that can significantly affect the number of messages sent across the network. Block fetch is used only with cursors that will not update or delete data. With block fetch, DB2 groups the rows retrieved by an SQL query into as large a “block” of rows as will fit in a message buffer, and transmits it over the network without requiring a message for every row.

DB2 can use two different types of block fetch:

- *Limited block fetch*
- *Continuous block fetch*

Both types of block fetch are used for both DRDA and private protocol, but the implementation of continuous block fetch for DRDA is slightly different than that for private protocol.

***Continuous block fetch:*** In terms of response times, the continuous block method is more efficient for larger result sets than the limited block method because fewer messages are transmitted and because overlapped processing is performed at the requester and server. But the continuous block method also uses more networking resources. Switching from continuous block to limited block allows applications to run when resources are critical.

The requester can use both forms of blocking, which can be in use at the same time with different servers.

If an application is doing read-only processing and can use continuous block fetch, the sequence goes like this:

1. A sends a message to open a cursor and begin fetching the block of rows at B.
2. B sends back a block of rows and A begins processing the first row.
3. B continues to send blocks of rows to A without further prompting. A processes the second and later rows as usual, but fetches them from a buffer on system A.

For private protocol, continuous block fetch uses one conversation for each open cursor. Having a dedicated conversation for each cursor allows the server to continue sending until all the rows are returned.

For DRDA, there is only one conversation, which must be made available to other SQL in the application. Thus, the server usually sends back a subset of all the rows. The number of rows that the server sends depends on the following factors:

- The size of each row
- The number of extra blocks that are requested by the requesting system versus the number of extra blocks the server will return

  For a DB2 for OS/390 and z/OS requester, the EXTRA BLOCKS REQ field on installation panel DSNTIP5 determines the maximum number of extra blocks requested. For a DB2 for OS/390 and z/OS server, the EXTRA BLOCKS SRV field on installation panel DSNTIP5 determines the maximum number of extra blocks requested.
- Whether continuous block fetch is enabled, and the number of extra rows that the server can return if it regulates that number

  To enable continuous block fetch for DRDA and to regulate the number of extra rows sent by a DB2 for OS/390 and z/OS server, you must use the OPTIMIZE FOR n ROWS clause on your SELECT statement. See “Optimizing for very large results sets for DRDA” on page 863 for more information.

If you want to use continuous block fetch for DRDA, it is recommended that the application fetch all the rows of the cursor before doing any other SQL. Fetching all the rows first, prevents the requester from having to buffer the data, which can consume a lot of storage. Choose carefully which applications should use continuous block fetch for DRDA.

***Limited block fetch:*** Limited block fetch guarantees the transfer of a minimum amount of data in response to each request from the requesting system. In the limited block method, a single conversation is used to transfer messages and data between the requester and server for multiple cursors. Processing at the requester and server is synchronous. The requester sends a request to the server, which causes the server to send a response back to the requester. The server must then wait for another request to tell it what should be done next.

***Block fetch with scrollable cursors:*** When a DB2 for OS/390 and z/OS requester uses a scrollable cursor to retrieve data from a DB2 for OS/390 and z/OS server, the following conditions are true:

- The requester never requests more than 64 rows in a query block, even if more rows fit in the query block. In addition, the requester never requests extra query blocks. This is true even if the setting of field EXTRA BLOCKS REQ in the DISTRIBUTED DATA FACILITY PANEL 2 installation panel on the requester allows extra query blocks to be requested. If you want to limit the number of rows that the server returns to fewer than 64, you can specify the FETCH FIRST *n* ROWS ONLY clause when you declare the cursor.
- The requester discards rows of the result table if the application does not use those rows. For example, if the application fetches row *n* and then fetches row *n*+2, the requester discards row *n*+1. The application gets better performance for a blocked scrollable cursor if it mostly scrolls forward, fetches most of the rows in a query block, and avoids frequent switching between FETCH ABSOLUTE statements with negative and positive values.
- If the scrollable cursor does not use block fetch, the server returns one row for each FETCH statement.

***LOB data and its effect on block fetch:*** For a non-scrollable blocked cursor, the server sends all the non-LOB data columns for a block of rows in one message, including LOB locator values. As each row is fetched by the application, the requester obtains the non-LOB data columns directly from the query block. If there are non-null and non-zero length LOB values in the row, those values are retrieved from the server at that time. This behavior limits the impact to the network by pacing the amount of data that is returned at any one time. If all LOB data columns are retrieved into LOB locator host variables or if the row does not contain any non-null or non-zero length LOB columns, then the whole row can be retrieved directly from the query block.

For a scrollable blocked cursor, the LOB data columns are returned at the same time as the non-LOB data columns. When the application fetches a row that is in the block, a separate message is not required to get the LOB columns.

***Ensuring block fetch:***

**General-use Programming Interface**

To use either limited or continuous block fetch, DB2 must determine that the cursor is not used for updating or deleting. The easiest way to indicate that the cursor does not modify data is to add the FOR FETCH ONLY or FOR READ ONLY clause to the query in the DECLARE CURSOR statement as in the following example:

```
EXEC SQL
  DECLARE THISEMP CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT, JOB
    FROM DSN8710.EMP
    WHERE WORKDEPT = 'D11'
    FOR FETCH ONLY
END-EXEC.
```

If you do not use FOR FETCH ONLY or FOR READ ONLY, DB2 still uses block fetch for the query if:

- The cursor is a non-scrollable cursor, and the result table of the cursor is read-only. This applies to static and dynamic cursors except for read-only views. (See Chapter 5 of *DB2 SQL Reference* for information about declaring a cursor as read-only.)
- The cursor is a scrollable cursor that is declared as INSENSITIVE, and the result table of the cursor is read-only.
- The cursor is a scrollable cursor that is declared as SENSITIVE, the result table of the cursor is read-only, and the value of bind option CURRENTDATA is NO.
- The result table of the cursor is not read-only, but the cursor is ambiguous, and the value of bind option CURRENTDATA is NO. A cursor is ambiguous when:
  - It is not defined with the clauses FOR FETCH ONLY, FOR READ ONLY, or FOR UPDATE OF.
  - It is not defined on a read-only result table.
  - It is not the target of a WHERE CURRENT clause on an SQL UPDATE or DELETE statement.
  - It is in a plan or package that contains the SQL statements PREPARE or EXECUTE IMMEDIATE.

DB2 triggers block fetch for static SQL only when it can detect that no updates or deletes are in the application. For dynamic statements, because DB2 cannot detect what follows in the program, the decision to use block fetch is based on the declaration of the cursor.

DB2 does not use continuous block fetch if:

- The cursor is referred to in the statement DELETE WHERE CURRENT OF elsewhere in the program.
- The cursor statement appears that it can be updated at the requesting system. (DB2 does not check whether the cursor references a view at the server that cannot be updated.)

The following three tables summarize the conditions under which a DB2 server uses block fetch:

- Table 119 shows the conditions for a non-scrollable cursor.

*Table 119. Effect of CURRENTDATA and cursor type on block fetch for a non-scrollable cursor*

| Isolation | CURRENTDATA | Cursor type | Block fetch |
|---|---|---|---|
| CS, RR, or RS | Yes | Read-only | Yes |
| | | Updatable | No |
| | | Ambiguous | No |
| | No | Read-only | Yes |
| | | Updatable | No |
| | | Ambiguous | Yes |
| UR | Yes | Read-only | Yes |
| | No | Read-only | Yes |

- Table 120 on page 863 shows the conditions for a scrollable cursor that is not used to retrieve a stored procedure result set.

*Table 120. Effect of CURRENTDATA and isolation level on block fetch for a scrollable cursor that is not used for a stored procedure result set*

| Isolation | Cursor sensitivity | CURRENT–DATA | Cursor type | Block fetch |
|---|---|---|---|---|
| CS, RR, or RS | INSENSITIVE | Yes | Read-only | Yes |
| | | No | Read-only | Yes |
| | SENSITIVE | Yes | Read-only | No |
| | | | Updatable | No |
| | | | Ambiguous | No |
| | | No | Read-only | Yes |
| | | | Updatable | No |
| | | | Ambiguous | Yes |
| UR | INSENSITIVE | Yes | Read-only | Yes |
| | | No | Read-only | Yes |
| | SENSITIVE | Yes | Read-only | Yes |
| | | No | Read-only | Yes |

- Table 121 shows the conditions for a scrollable cursor that is used to retrieve a stored procedure result set.

*Table 121. Effect of CURRENTDATA and isolation level on block fetch for a scrollable cursor that is used for a stored procedure result set*

| Isolation | Cursor sensitivity | CURRENT–DATA | Cursor type | Block fetch |
|---|---|---|---|---|
| CS, RR, or RS | INSENSITIVE | Yes | Read-only | Yes |
| | | No | Read-only | Yes |
| | SENSITIVE | Yes | Read-only | No |
| | | No | Read-only | Yes |
| UR | INSENSITIVE | Yes | Read-only | Yes |
| | | No | Read-only | Yes |
| | SENSITIVE | Yes | Read-only | Yes |
| | | No | Read-only | Yes |

**End of General-use Programming Interface**

## Optimizing for very large results sets for DRDA

Enabling a DB2 client to request multiple query blocks on each transmission can reduce network activity and improve performance significantly for applications that use DRDA access to download large amounts of data. You can specify a large value of *n* in the OPTIMIZE FOR *n* ROWS clause of a SELECT statement to increase the number of DRDA query blocks that a DB2 server returns in each network transmission for a non-scrollable cursor. If *n* is greater than the number of rows that fit in a DRDA query block, OPTIMIZE FOR *n* ROWS lets the DRDA client request multiple blocks of query data on each network transmission instead of requesting a new block when the first block is full. This use of OPTIMIZE FOR *n* ROWS is intended only for applications in which the application opens a cursor and downloads great amounts of data. The OPTIMIZE FOR *n* ROWS clause has no effect on scrollable cursors.

**Recommendation:** Because there is only one conversation used by the application's SQL, do not try to do other SQL work until the entire answer set is processed. If the requester issues another SQL statement before the previous statement's answer set has been received off the network, DDF must buffer them in its address space. Up to 10 MB can be buffered in this way.

Because specifying a large number of network blocks can saturate the network, limit the number of blocks according to what your network can handle. You can limit the number of blocks used for these large download operations. When the client supports extra query blocks, DB2 chooses the *smallest* of the following values when determining the number of query blocks to send:

- The number of blocks into which the number of rows (*n*) on the OPTIMIZE clause will fit. For example, assume you specify 10000 rows for *n*, and the size of each row that is returned is approximately 100 bytes. If the block size used is 32 KB (32768 bytes), the calculation is as follows:
  ```
  (10000 * 100) / 32768 = 31 blocks
  ```
- The DB2 server value for the installation option EXTRA BLOCKS SRV install option on panel DSNTIP5. The maximum value that you can specify is 100.
- The client's extra query block limit, which is obtained from the DRDA MAXBLKEXT parameter received from the client. When DB2 for OS/390 and z/OS acts as a DRDA client, you set this parameter at installation time with the EXTRA BLOCKS REQ option of the DSNTIP5 panel. The maximum value that you can specify is 100.

If the client does not support extra query blocks, the DB2 server on OS/390 or z/OS automatically reduces the value of *n* to match the number of rows that fit within a DRDA query block.

**Recommendation for cursors that are defined WITH HOLD:** Do not set a large number of query blocks for cursors that are defined WITH HOLD. If the application commits while there are still a lot of blocks in the network, DB2 buffers the blocks in the requester's memory (the *ssnm*DIST address space if the requester is a DB2 for OS/390 and z/OS) before the commit can be sent to the server.

For examples of performance problems that can occur from not using OPTIMIZE FOR *n* ROWS when downloading large amounts of data, see Part 4 of *DB2 Application Programming and SQL Guide*.

## Optimizing for small results sets for DRDA

When a client does not need all the rows from a potentially large result set, preventing the DB2 server from returning all the rows for a query can reduce network activity and improve performance significantly for DRDA applications. You can use either the OPTIMIZE FOR *n* ROWS clause or the FETCH FIRST *n* ROWS ONLY clause of a SELECT statement to limit the number of rows returned to a client program.

**Using OPTIMIZE FOR *n* ROWS:** When you specify OPTIMIZE FOR *n* ROWS and *n* is less than the number of rows that fit in the DRDA query block (default size on OS/390 or z/OS is 32 KB), the DB2 server prefetches and returns only as many rows as fit into the query block. For example, if the client application is interested in seeing only one screen of data, specify OPTIMIZE FOR *n* ROWS, choosing a small number for *n*, such as 3 or 4. The OPTIMIZE FOR *n* ROWS clause has no effect on scrollable cursors.

**Using FETCH FIRST *n* ROWS ONLY:** When you specify FETCH FIRST *n* ROWS ONLY, the DB2 server prefetches and returns only *n* rows even if more rows can fit into the DRDA query block. FETCH FIRST *n* ROWS ONLY can prevent the prefetching of any unnecessary rows. For example, if you need only one row of the result table, FETCH FIRST 1 ROW ONLY causes only one row to be prefetched and returned. Had you specified only OPTIMIZE FOR 1 ROW, enough rows to fit into the query block would have been prefetched and returned.

If you specify FETCH FIRST *n* ROWS ONLY, then OPTIMIZE FOR *n* ROWS is implied, and DB2 optimizes the query as if you had specified OPTIMIZE FOR *n* ROWS. If you specify both clauses, DB2 optimizes the query as if you had specified OPTIMIZE FOR *n* ROWS, where *n* is the lesser of the values specified for each clause.

When you use FETCH FIRST *n* ROWS ONLY, DB2 might use a *fast implicit close.* Fast implicit close means that during a distributed query, the DB2 server automatically closes the cursor when it prefetches the *n*th row if FETCH FIRST *n* ROWS ONLY is specified or when there are no more rows to return. Fast implicit close can improve performance because it can save an additional network transmission between the client and the server.

DB2 uses fast implicit close when the following conditions are true:

- The query uses limited block fetch.
- The query retrieves no LOBs.
- The cursor is not a scrollable cursor.
- Either of the following conditions is true:
  - The cursor is declared WITH HOLD, and the package or plan that contains the cursor is bound with the KEEPDYNAMIC(YES) option.
  - The cursor is not defined WITH HOLD.

When you use FETCH FIRST *n* ROWS ONLY and DB2 does a fast implicit close, the DB2 server closes the cursor after it prefetches *n* rows, or when there are no more rows.

## The serving system

For access using DB2 private protocol, the serving system is the DB2 system on which the SQL is dynamically executed. For access using DRDA, the serving system is the system on which your remotely bound package executes.

If you are executing a package on a remote DBMS, then improving performance on the server depends on the nature of the server. If the remote DBMS on which the package executes is another DB2, then the information in “Chapter 33. Using EXPLAIN to improve SQL performance” on page 789 is appropriate for access path considerations.

Other considerations that could affect performance on a DB2 server are:

- The maximum number of database access threads that the server allows to be allocated concurrently. (This is the MAX REMOTE ACTIVE option on installation panel DSNTIPE.) A request can be queued while waiting for an available thread. Making sure that requesters commit frequently can let threads be used by other requesters. See “Setting thread limits for database access threads” on page 625 for more information.

- The priority of database access threads on the remote system. A low priority could impede your application's distributed performance. See "Using Workload Manager to set performance objectives" on page 629 for more information.
- For instructions on avoiding RACF calls at the server, see "Controlling requests from remote applications" on page 176, and more particularly "Do you manage inbound IDs through DB2 or RACF?" on page 181.

When DB2 is the server, it is a good idea to activate accounting trace class 7. This provides accounting information at the package level, which can be very useful in determining performance problems.

## Monitoring DB2 in a distributed environment

DB2 provides several ways to monitor DB2 data and events in a distributed environment. You can use the DISPLAY command and the trace facility to obtain information. DB2 can also return server-elapsed time to certain types of client applications.

### Using the DISPLAY command

The DB2 DISPLAY command gives you information about the status of threads, databases, tracing, allied subsystems, and applications. Several forms of the DISPLAY command are particularly helpful for monitoring DB2: DISPLAY THREAD, DISPLAY LOCATION, DISPLAY DATABASE, and DISPLAY TRACE. For the detailed syntax of each command, refer to Chapter 2 of *DB2 Command Reference*. See also:

"Monitoring threads" on page 283
"The command DISPLAY LOCATION" on page 311

### Tracing distributed events

A number of IFCIDs, including IFCID 0001 (statistics) and IFCID 0003 (accounting), record distributed data and events.

If your applications update data at other sites, turn on the statistics class 4 trace and always keep it active. This statistics trace covers error situations surrounding in doubt threads; it provides a history of events that might impact data availability and data consistency.

DB2 accounting records are created separately at the requester and each server. Events are recorded in the accounting record at the location where they occur. When a thread becomes active, the accounting fields are reset. Later, when the thread becomes inactive or is terminated, the accounting record is created.

Figure 121 on page 867 shows the relationship of the accounting class 1 and 2 times and the requester and server accounting records. Figure 122 on page 868 and Figure 123 on page 869 show the server and requester distributed data facility blocks from the DB2 PM accounting long trace.

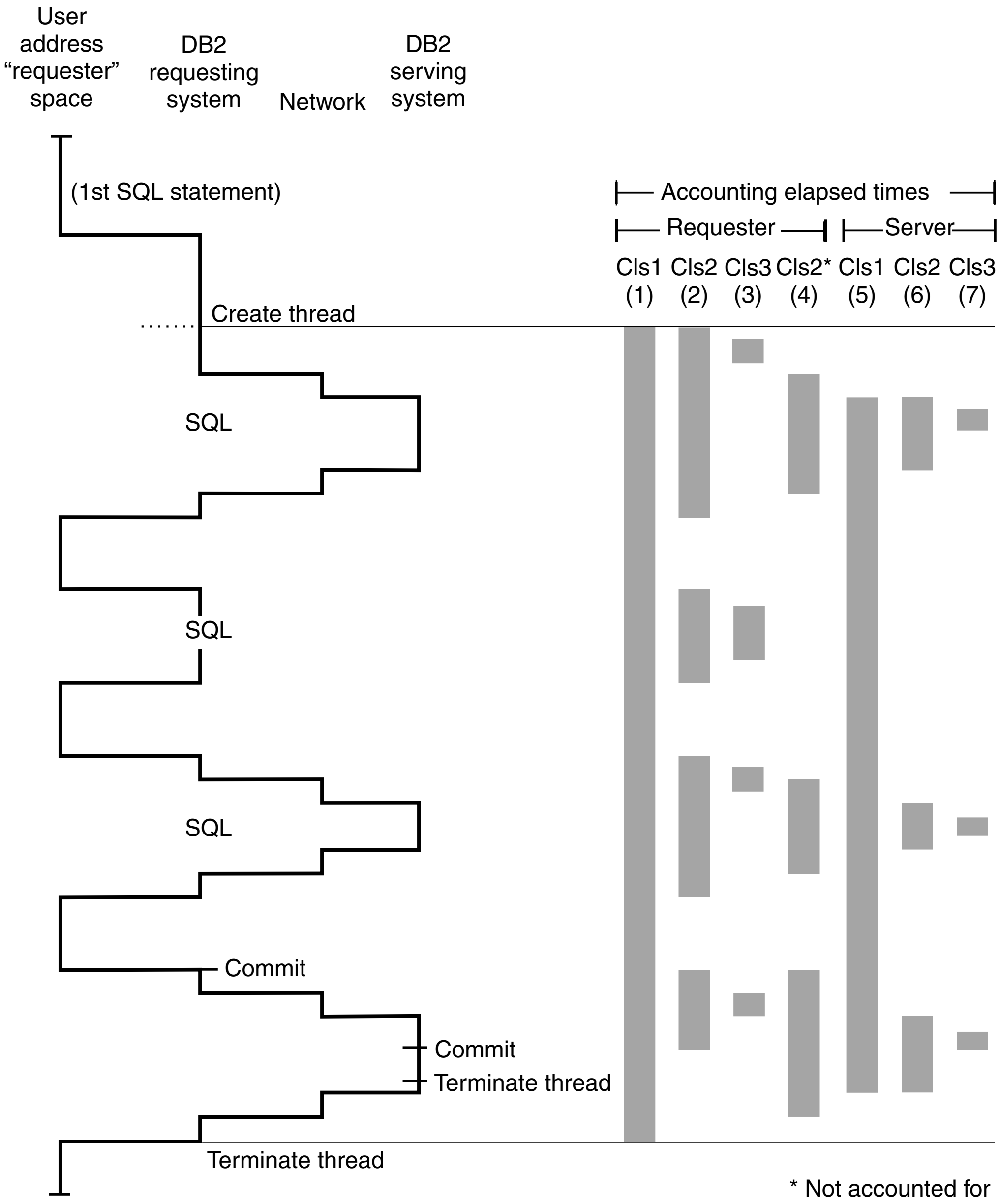


*Figure 121. Elapsed times in a DDF environment as reported by DB2 PM. These times are valid for access that uses either DRDA or private protocol (except as noted).*

This figure is a very simplified picture of the processes that go on in the serving system. It does not show block fetch statements and is only applicable to a single row retrieval.

The various elapsed times referred to in the header are:

- (1) - Requester Cls1

  This time is reported in the ELAPSED TIME field under the APPL (CLASS 1) column near the top of the DB2 PM accounting long trace for the requesting DB2 subsystem. It represents the elapsed time from the creation of the allied distributed thread until the termination of the allied distributed thread.
- (2) - Requester Cls2

  This time is reported in the ELAPSED TIME field under the DB2 (CLASS 2) column near the top of the DB2 PM accounting long trace for the requesting DB2 subsystem. It represents the elapsed time from when the application passed the SQL statements to the local DB2 system until return. This is considered “In DB2” time.

- (3) - Requester Cls3

  This time is reported in the TOTAL CLASS 3 field under the CLASS 3 SUSP column near the top of the DB2 PM accounting long trace for the requesting DB2 system. It represents the amount of time the requesting DB2 system spent suspended waiting for locks or I/O.

- (4) - Requester Cls2* (Requester wait time for activities not in DB2)

  This time is reported in the NOT ACCOUNT field of the DB2 PM accounting report for the requesting DB2 subsystem. It represents the time the requester spent waiting for the network and server to process the request. It is not actually time spent in DB2.

- (5) - Server Cls1

  This time is reported in the ELAPSED TIME field under the APPL (CLASS 1) column near the top of the DB2 PM accounting long trace for the serving DB2 subsystem. It represents the elapsed time from the creation of the database access thread until the termination of the database access thread.

- (6) - Server Cls2

  This time is reported in the ELAPSED TIME field under the DB2 (CLASS 2) column near the top of the DB2 PM accounting long trace of the serving DB2 subsystem. It represents the elapsed time to process the SQL statements and the commit at the server.

- (7) - Server Cls3

  This time is reported in the TOTAL CLASS 3 field under the CLASS 3 SUSP column near the top of the DB2 PM accounting long trace for the serving DB2 subsystem. It represents the amount of time the serving DB2 system spent suspended waiting for locks or I/O.

The Class 2 processing time (the TCB time) at the requester does not include processing time at the server. To determine the total Class 2 processing time, add the Class 2 time at the requester to the Class 2 time at the server.

Likewise, add the getpage counts, prefetch counts, locking counts, and I/O counts of the requester to the equivalent counts at the server. For private protocol, SQL activity is counted at both the requester and server. For DRDA, SQL activity is counted only at the server.

```
---- DISTRIBUTED ACTIVITY -------------------------------------------------------------------------------------
SERVER                 : BOEBDB2SERV      SUCCESSFULLY ALLOC.CONV: C N/A      MSG.IN BUFFER  E :      0
PRODUCT ID             : DB2              CONVERSATION TERMINATED:   N/A
PRODUCT VERSION        : V6 R1 M0         MAX OPEN CONVERSATIONS :   N/A      PREPARE SENT     :      1
METHOD                 : DRDA PROTOCOL    CONT->LIM.BL.FTCH SWCH : D N/A      LASTAGN.SENT     :      0
REQUESTER ELAP.TIME    :   0.685629                                           MESSAGES SENT    :      3
SERVER ELAPSED TIME    :        N/A       COMMIT(2) RESP.RECV.   :     1      MESSAGES RECEIVED:      2
SERVER CPU TIME        :        N/A       BACKOUT(2) RESP.RECV.  :     0      BYTES SENT       :   9416
DBAT WAITING TIME      :   0.026118       TRANSACT.SENT          :     1      BYTES RECEIVED   :   1497
COMMIT (2) SENT        :          1       COMMT(1)SENT           :     0      BLOCKS RECEIVED  :      0
BACKOUT(2) SENT        :          0       ROLLB(1)SENT           :     0      STMT BOUND AT SER: F  N/A
CONVERSATIONS INITIATED: A        1       SQL SENT               :     0
CONVERSATIONS QUEUED   : B        0       ROWS RECEIVED          :     1      FORGET RECEIVED  :      0
```

Figure 122. DDF block of a requester thread from a DB2 PM accounting long trace

```
---- DISTRIBUTED ACTIVITY -----------------------------------------------------------------------------------------
REQUESTER          : BOEBDB2REQU        ROLLBK(1) RECEIVED :     0      PREPARE RECEIVED   :     1
PRODUCT ID         : DB2                SQL RECEIVED       :     0      LAST AGENT RECV.   :     1
PRODUCT VERSION    : V6 R1 M0           COMMIT(2) RESP.SENT:     1      THREADS INDOUBT    :     0
METHOD             : DRDA PROTOCOL      BACKOUT(2)RESP.SENT:     0      MESSAGES.IN BUFFER :     0
COMMIT(2) RECEIVED :                1   BACKOUT(2)PERFORMED:     0      ROWS SENT          :     0
BACKOUT(2) RECEIVED:                0   MESSAGES SENT      :     3      BLOCKS SENT        :     0
COMMIT(2) PERFORMED:                1   MESSAGES RECEIVED  :     5      CONVERSAT.INITIATED:     1
TRANSACTIONS RECV. :                1   BYTES SENT         :   643      FORGET SENT        :     0
COMMIT(1) RECEIVED :                0   BYTES RECEIVED     :  3507
```

*Figure 123. DDF block of a server thread from a DB2 PM accounting long trace*

The accounting distributed fields for each serving or requesting location are collected from the viewpoint of this thread communicating with the other location identified. For example, SQL *sent* from the requester is SQL *received* at the server. Do not add together the distributed fields from the requester and the server.

Several fields in the distributed section merit specific attention. The number of conversations is reported in several fields:

- The number of conversation allocations is reported as CONVERSATIONS INITIATED ( A ).
- The number of conversation requests queued during allocation is reported as CONVERSATIONS QUEUED ( B ).
- The number of successful conversation allocations is reported as SUCCESSFULLY ALLOC.CONV ( C ).
- The number of times a switch was made from continuous block fetch to limited block fetch is reported as CONT->LIM.BL.FTCH ( D ). This is only applicable to access that uses DB2 private protocol.

You can use the difference between initiated allocations and successful allocations to identify a session resource constraint problem. If the number of conversations queued is high, or if the number of times a switch was made from continuous to limited block fetch is high, you might want to tune VTAM to increase the number of conversations. VTAM and network parameter definitions are important factors in the performance of DB2 distributed processing. For more information, see *VTAM for MVS/ESA Network Implementation Guide*.

Bytes sent, bytes received, messages sent, and messages received are recorded at both the requester and the server. They provide information on the volume of data transmitted. However, because of the way distributed SQL is processed for private protocol, more bytes may be reported as sent than are reported as received.

To determine the percentage of the rows transmitted by block fetch, compare the total number of rows sent to the number of rows sent in a block fetch buffer, which is reported as MSG.IN BUFFER ( E ). The number of rows sent is reported at the server, and the number of rows received is reported at the requester. Block fetch can significantly affect the number of rows sent across the network.

The number of SQL statements bound for remote access is the number of statements dynamically bound at the server for private protocol. This field is maintained at the requester and is reported as STMT BOUND AT SER ( F ).

Because of the manner in which distributed SQL is processed, there may be a small difference in the number of rows reported as sent versus received. However, a significantly lower number of rows received may indicate that the application did not fetch the entire answer set. This is especially true for access that uses DB2 private protocol.

## Reporting server-elapsed time

Client applications that access DB2 data using DRDA access can request that DB2 return the server-elapsed time. server-elapsed time allows remote clients to determine the actual amount of time it takes for DB2 to parse a remote request, process any SQL statements required to satisfy the request, and generate the reply. Because server-elapsed time does not include any of the network time used to receive the request or send the reply, client applications can use the information to quickly isolate poor response times to the network or to the DB2 server without you having to perform traces on the server. Only application clients using DB2 Connect Version 7 can request the server-elapsed time. When the System Monitor statement switch has been turned on, DB2 returns server-elapsed time information as a new element through the regular Snapshot Monitoring APIs.

# Using RMF to monitor distributed processing

If you use RMF to monitor DDF work, it is important to understand how DDF is using the enclave SRBs described in “Using Workload Manager to set performance objectives” on page 629. The information that is reported using RMF or an equivalent product in the SMF 72 records are the portions of the client’s request that are covered by individual enclaves. The way DDF uses enclaves relates directly to whether the DDF thread can become inactive.

## Duration of an enclave

“Using inactive threads” on page 626 describes the difference between threads that are always active and those that can become inactive (sometimes active threads). From an MVS enclave point of view, an enclave only lasts as long as the thread is active. Any inactive period, such as think time, is not using an enclave and is not managed by MVS’s SRM. Inactive periods are therefore not reported in the SMF 72 record.

Active threads that cannot become inactive (*always active* threads) are treated as a single enclave from the time it is created until the time it is terminated. This means that the entire life of the database access thread is reported in the SMF 72 record, regardless of whether SQL work is actually being processed. Figure 124 on page 871 contrasts the two types of threads and their management by SRM.

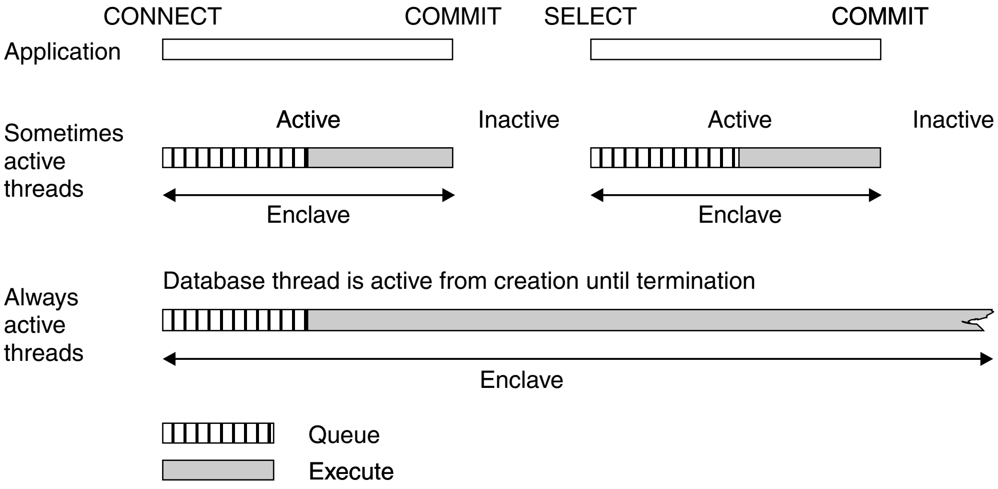


*Figure 124. Contrasting ’always active’ vs. ’sometimes active’ threads*

**Queue Time:** Note that the information reported back to RMF includes queue time. This particular queue time includes waiting for a new or existing thread to become available. This queue time is also reported in DB2 class 3 times, but class 3 times also include time waiting for locks or I/O after the thread is processing work.

## RMF records for enclaves

The two most frequently used SMF records are types 30 and 72. The type 30 record contains resource consumption at the address space level. You can pull out total enclave usage from the record, but you must use DB2 accounting traces to see resource consumption for a particular enclave.

Type 72 records contain data collected by RMF monitor 1. There is one type 72 record for each service class period, report class, performance group number (PGN) period, and report performance group (RPGN) per RMF monitor 1 interval. Each enclave contributes its data to one type 72 for the service class or PGN and to zero or one (0 or 1) type 72 records for the report class or RPGN. By using WLM classification rules, you can segregate enclaves into different service classes or report classes (or PGNs or RPGNs, if using compatibility mode). By doing this, you can understand the DDF work better.

# Chapter 36. Monitoring and tuning stored procedures and user-defined functions

Table 122 summarizes the differences between stored procedures that run in WLM-established stored procedures address spaces and those that run in DB2-established stored procedure address space. User-defined functions must run in a WLM-established address space. Performance tuning information for user-defined functions and for stored procedures in a WLM-established address space is the same.

*Table 122. Comparing WLM-established and DB2-established stored procedures*

| DB2-established | WLM-established | More information |
|---|---|---|
| Use a single address space for stored procedures:<br>• A failure in one stored procedure can affect other stored procedures that are running in that address space.<br>• Can be difficult to support more than 50 stored procedures running at the same time because of storage that language products need below the 16MB line. | Use many address spaces for stored procedures and user-defined functions:<br>• Possible to isolate procedures and functions from one another so that failures do not affect others that are running in other address spaces.<br>• Reduces demand for storage below the 16MB line and thereby removes the limitation on the number of procedures and functions that can run concurrently.<br>• Only one utility can be invoked by a stored procedure in one address space at any given time. The start parameter NUMTCB on the WLM Application-Environment panel has to be set to 1. | "Controlling address space storage" on page 874, and Figure 125 on page 876 |
| Incoming requests for stored procedures are handled in a first-in, first-out order. | Requests are handled in priority order. | "Using Workload Manager to set performance objectives" on page 629 |
| Stored procedures run at the priority of the stored procedures address space. | Stored procedures inherit the MVS dispatching priority of the DB2 thread that issues the CALL statement. User-defined functions inherit the priority of the DB2 thread that invoked the procedure. | "Using Workload Manager to set performance objectives" on page 629 |
| No ability to customize the environment. | Each address space is associated with a WLM *application environment* that you specify. An application environment is an attribute that you associate on the CREATE statement for the function or procedure. The environment determines which JCL procedure is used to run a particular stored procedure. | "Assigning procedures and functions to WLM application environments" on page 875 |
| Must run as a MAIN program. | Can run as a MAIN or SUB program. SUB programs can run significantly faster, but the subprogram must do more initialization and cleanup processing itself rather than relying on LE/370 to handle that. | Part 6 of *DB2 Application Programming and SQL Guide* |

*Table 122. Comparing WLM-established and DB2-established stored procedures (continued)*

| DB2-established | WLM-established | More information |
|---|---|---|
| You can access non-relational data, but that data is not included in your SQL unit of work. It is a separate unit of work. | You can access non-relational data. If the non-relational data is managed by OS/390 RRS, the updates to that data are part of your SQL unit of work. | Part 6 of *DB2 Application Programming and SQL Guide* |
| Stored procedures access protected MVS resources with the authority of the stored procedures address space. | Procedures or functions can access protected MVS resources with one of three authorities, as specified on the SECURITY option of the CREATE FUNCTION or CREATE PROCEDURE statement:<br>• The authority of the WLM-established address space (SECURITY=DB2)<br>• The authority of the invoker of the stored procedure or user-defined function (SECURITY=USER)<br>• The authority of the definer of the stored procedure or user-defined function (SECURITY=DEFINER) | Part 3 (Volume 1) of *DB2 Administration Guide* |

## Controlling address space storage

To maximize the number of procedures or functions that can run concurrently in an address space, use the following guidelines:

- Set REGION size for the address spaces to REGION=0 to obtain the largest possible amount of storage below the 16MB line.
- Limit storage required by application programs below the 16MB line by:
  - Linking editing programs above the line with AMODE(31) and RMODE(ANY) attributes
  - Using the RES and DATA(31) compiler options for COBOL programs
- Limit storage required by Language Environment by using these runtime options:
  - HEAP(,,ANY) to allocate program heap storage above the 16MB line
  - STACK(,,ANY,) to allocate program stack storage above the 16MB line
  - STORAGE(,,,4K) to reduce reserve storage area below the line to 4KB
  - BELOWHEAP(4K,,) to reduce the heap storage below the line to 4KB
  - LIBSTACK(4K,,) to reduce the library stack below the line to 4KB
  - ALL31(ON) to indicate all programs contained in the stored procedure run with AMODE(31) and RMODE(ANY)

If you follow these guidelines, each TCB that runs in the DB2-established stored procedures address space requires approximately 100 KB below the 16MB line. Each TCB that runs in a WLM-established stored procedures address space uses approximately 200 KB below the line, but because you can have fewer stored procedures (and user-defined functions) per address space, your total below-the-line cost can be less.

DB2 needs extra storage for stored procedures and user-defined functions in the WLM-established address space because you can create both main and sub programs, and DB2 must create an environment for each.

A stored procedure can invoke only one utility in one address space at any given time because of the resource requirements of utilities. On the WLM Application-Environment panel, set NUMTCB to 1. See Figure 125 on page 876. However, a stored procedure can invoke several compatible utilities at the same time if you create multiple WLM address spaces and direct each utility to a different address space.

***Dynamically extending load libraries:*** Use partitioned data set extended (PDSEs) for load libraries containing stored procedures. Using PDSEs may eliminate your need to stop and start the stored procedures address space due to growth of the load libraries. If a load library grows from additions or replacements, the library may have to be extended.

If you use PDSEs for the load libraries, the new extent information is dynamically updated and you do not need to stop and start the address space. If PDSs are used, load failures may occur because the new extent information is not available.

## Assigning procedures and functions to WLM application environments

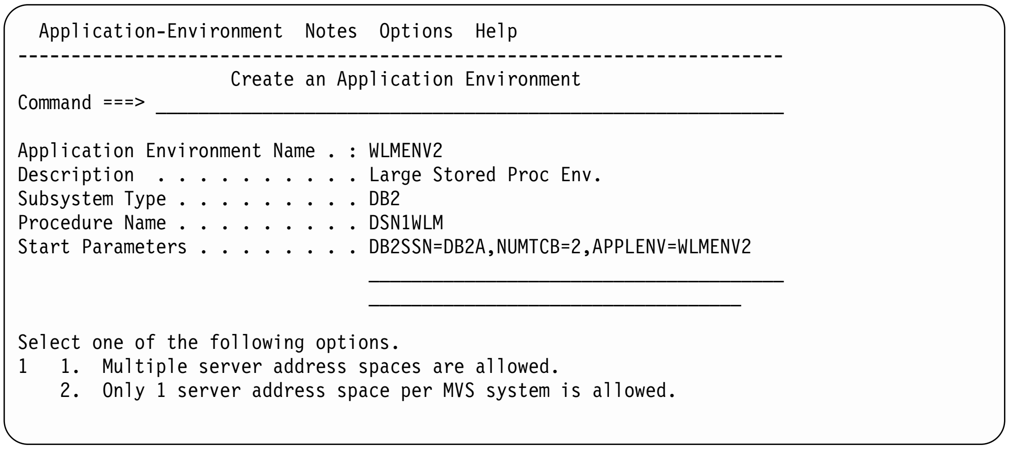
Workload manager routes work to address spaces based on the application environment name and service class associated with the stored procedure or function. You must use WLM panels to associate an application environment name with the JCL procedure used to start an address space. See *OS/390 MVS Planning: Workload Management* for details about workload management panels.

Other tasks must be completed before a stored procedure or user-defined function can run in a WLM-established stored procedures address space. Here is a summary of those tasks:

1. Make sure you have a numeric value specified in the TIMEOUT VALUE field of installation panel DSNTIPX. If you have problems with setting up the environment, this timeout value ensures that your request to execute a stored procedure will not wait for an unlimited amount of time.
2. If you want to convert to WLM-managed stored procedures that use the DB2-established stored procedure address space (*ssnm*SPAS), you must link edit them or code them so that they use the Recoverable Resource Manager Services attachment facility (RRSAF) instead of the call attachment facility. Use the JCL startup procedure for WLM-established stored procedures address space that was created when you installed or migrated as a model. (The default name is *ssnm*WLM.)

   Unless a particular application environment or caller’s service class is not used for a long time, WLM creates on demand at least one address space for each combination of application environment name and caller’s service class that is encountered in the workload. For example, if there are five application environment names that each have calling threads with six possible service classes, and all those combinations are in demand, it is possible to have 30 address spaces containing stored procedures or user-defined functions.

   To prevent creating too many address spaces, create a relatively small number of WLM application environments and MVS service classes.
3. Use the WLM application environment panels to associate the environment name with the JCL procedure. Figure 125 on page 876 is an example of this panel.



```
  Application-Environment  Notes  Options  Help
------------------------------------------------------------------------
                    Create an Application Environment
Command ===> ___________________________________________________________

Application Environment Name . : WLMENV2
Description  . . . . . . . . . . Large Stored Proc Env.
Subsystem Type . . . . . . . . . DB2
Procedure Name . . . . . . . . . DSN1WLM
Start Parameters . . . . . . . . DB2SSN=DB2A,NUMTCB=2,APPLENV=WLMENV2
                                 _______________________________________
                                 ___________________________________

Select one of the following options.
1   1.  Multiple server address spaces are allowed.
    2.  Only 1 server address space per MVS system is allowed.
```

*Figure 125. WLM panel to create an application environment. You can also use the variable &IWMSSNM for the DB2SSN parameter (DB2SSN=&IWMSSNM). This variable represents the name of the subsystem for which you are starting this address space. This variable is useful for using the same JCL procedure for multiple DB2 subsystems.*

4. Specify the WLM application environment name for the WLM_ENVIRONMENT option on CREATE or ALTER PROCEDURE (or FUNCTION) to associate a stored procedure or user-defined function with an application environment.
5. Using the install utility in the WLM application, install the WLM service definition that contains information about this application environment into the couple data set.
6. Activate a WLM policy from the installed service definition.
7. Issue STOP PROCEDURE and START PROCEDURE for any stored procedures that run in the *ssnm*SPAS address space. This process allows those procedures to pick up the new value for WLM environment.
8. Begin running stored procedures.

## Providing DB2 cost information for accessing user-defined table functions

User-defined table functions add additional access cost to the execution of an SQL statement. For DB2 to factor in the effect of user-defined table functions in the selection of the best access path for an SQL statement, the total cost of the user-defined table function must be determined.

The total cost of a table function consists of the following three components:

- The initialization cost that results from the first call processing
- The cost that is associated with acquiring a single row
- The final call cost that performs the clean up processing

These costs, though, are not known to DB2 when I/O costs are added to the CPU cost.

To assist DB2 in determining the cost of user-defined table functions, you can use four fields in SYSIBM.SYSROUTINES. Use the following fields to provide cost information:

- IOS_PER_INVOC for the estimated number of I/Os per row
- INSTS_PER_INVOC for the estimated number of instructions

- INITIAL_IOS for the estimated number of I/Os performed the first and last time the function is invoked
- INITIAL_INSTS for the estimated number of instructions for the first and last time the function is invoked

These values, along with the CARDINALITY value of the table being accessed, are used by DB2 to determine the cost. The results of the calculations can influence such things as the join sequence for a multi-table join and the cost estimates generated for and used in predictive governing.

Determine values for the four fields by examining the source code for the table function. Estimate the I/Os by examining the code executed during the FIRST call and FINAL call. Look for the code executed during the OPEN, FETCH, and CLOSE calls. The costs for the OPEN and CLOSE calls can be amortized over the expected number of rows returned. Estimate the I/O cost by providing the number of I/Os that will be issued. Include the I/Os for any file access.

Figure the instruction cost by counting the number of high level instructions executed in the user-defined table function and multiplying it by a factor of 20. For assembler programs, the instruction cost is the number of assembler instructions.

If SQL statements are issued within the user-defined table function, use DB2 Estimator to determine the number of instructions and I/Os for the statements. Examining the JES job statistics for a batch program doing equivalent functions can also be helpful. For all fields, a precise number of instructions is not required. Because DB2 already accounts for the costs of invoking table functions, these costs should not be included in the estimates.

The following example shows how these fields can be updated. The authority to update is the same authority as that required to update any catalog statistics column.

```
UPDATE SYSIBM.SYSROUTINES SET
   IOS_PER_INVOC   = 0.0,
   INSTS_PER_INVOC = 4.5E3,
   INITIAL_IOS     = 2.0
   INITIAL_INSTS   = 1.0E4,
   CARDINALITY     = 5E3
WHERE
   SCHEMA = 'SYSADM' AND
   SPECIFICNAME = 'FUNCTION1' AND
   ROUTINETYPE = 'F';
```

## Accounting trace

Through a stored procedure one SQL statement generates other SQL statements under the same thread. The processing done by the stored procedure is included in DB2’s class 1 and class 2 times for accounting.

The accounting report on the server has several fields that specifically relate to stored procedures processing, as shown in Figure 126 on page 878.

```
PLANNAME: PU22301

 AVERAGE       APPL (CLASS 1)  DB2 (CLASS 2)   IFI (CLASS 5)    CLASS 3 SUSP.   AVERAGE TIME  AV.EVENT
 ------------  --------------  --------------  --------------   --------------  ------------  --------
 ELAPSED TIME        5.773449        3.619543             N/P   LOCK/LATCH          1.500181      1.09
  NON-NESTED         2.014711        1.533210             N/A   SYNCHRON. I/O       0.002096      0.13
  STORED PROC A      3.758738        2.086333             N/A    DATABASE I/O       0.000810      0.09
  UDF                0.000000        0.000000             N/A    LOG WRITE I/O      0.001286      0.04
  TRIGGER            0.000000        0.000000             N/A   OTHER READ I/O      0.000000      0.00
                                                                OTHER WRTE I/O      0.000000      0.00
 CPU TIME            0.141721        0.093469             N/P   SER.TASK SWTCH      0.860814      1.04
  AGENT              0.141721        0.093469             N/P    UPDATE COMMIT      0.010989      0.06
   NON-NESTED        0.048918        0.004176             N/A    OPEN/CLOSE         0.448021      0.20
   STORED PROC       0.092802        0.089294             N/A    SYSLGRNG REC       0.193708      0.61
   UDF               0.000000        0.000000             N/A    EXT/DEL/DEF        0.160772      0.01
   TRIGGER           0.000000        0.000000             N/A    OTHER  SERVICE     0.047324      0.16
  PAR.TASKS          0.000000        0.000000             N/A   ARC.LOG(QUIES)      0.000000      0.00
                                                                ARC.LOG READ        0.000000      0.00
 SUSPEND TIME             N/A        2.832920             N/A   STORED PROC. B      0.129187      0.04
  AGENT                   N/A        2.832920             N/A   UDF SCHEDULE        0.000000      0.00
  PAR.TASKS               N/A        0.000000             N/A   DRAIN LOCK          0.000000      0.00
                                                                CLAIM RELEASE       0.000000      0.00
 NOT ACCOUNT.             N/A        0.693154             N/A   PAGE LATCH          0.000000      0.00
 DB2 ENT/EXIT             N/A            8.96             N/A   NOTIFY MSGS.        0.000000      0.00
 EN/EX-STPROC             N/A           41.74             N/A   GLOBAL CONT.        0.340642      7.37
 EN/EX-UDF                N/A             N/A             N/P   TOTAL CLASS 3       2.832920      9.67
 DCAPT.DESCR.             N/A             N/A             N/P
 LOG EXTRACT.             N/A             N/A             N/P

⋮

STORED PROCEDURES   AVERAGE     TOTAL
-----------------  --------  --------
CALL STATEMENTS C      1.00         1
PROCEDURE ABENDS       0.00         0
CALL TIMEOUT    D      0.00         0
CALL REJECT            0.00         0

⋮
```

*Figure 126. Partial long accounting report, server —stored procedures*

***Descriptions of fields:***

- The part of the total CPU time that was spent satisfying stored procedures requests is indicated in A.
- The amount of time spent waiting for a stored procedure to be scheduled is indicated in B.
- The number of calls to stored procedures is indicated in C.
- The number of times a stored procedure timed out waiting to be scheduled is shown in D.

***What to do for excessive timeouts or wait time:*** If you have excessive wait time (B) or timeouts (D), there are several possible causes.

For user-defined functions, or for stored procedures in a **WLM-established** address space, the causes for excessive wait time include:

- The priority of the service class that is running the stored procedure is not high enough.
- You are running in compatibility mode, which means you might have to manually start more address spaces.
- If you are using goal mode, make sure that the application environment is available by using the MVS command DISPLAY WLM,APPLENV=*applenv*. If the

application environment is quiesced, WLM does not start any address spaces for that environment; CALL statements are queued or rejected.

For stored procedures in a **DB2-established** address space, the causes for excessive wait time include:

- Someone issued the DB2 command STOP PROCEDURE ACTION(QUEUE) that caused requests to queue up for a long time and time out.
- The stored procedures are hanging onto the *ssnm*SPAS TCBs for too long. In this case, you need to find out why this is happening.

  If you are getting many DB2 lock suspensions, maybe you have too many *ssnm*SPAS TCBs, causing them to encounter too many lock conflicts with one another. Or, maybe you just need to make code changes to your application. Or, you might need to change your database design to reduce the number of lock suspensions.
- If the stored procedures are getting in and out quickly, maybe you don't have enough *ssnm*SPAS TCBs to handle the work load. In this case, increase the number on field NUMBER OF TCBS on installation panel DSNTIPX.

## Accounting for nested activities

The accounting class 1 and class 2 CPU and elapsed times for triggers, stored procedures, and user-defined functions are accumulated in separate fields and exclude any time accumulated in other nested activity. These CPU and elapsed times are accumulated for each category during the execution of each agent until agent deallocation. Package accounting can be used to break out accounting data for execution of individual stored procedures, user-defined functions, or triggers. The following sample (Figure 127) shows an agent that executes multiple types of DB2 nested activities.

```
Time    Application       DB2                   SP             UDF
------  ---------------   ------------------    ------------   ------------
 T0     Code
 T1     SQL-------------->
 T2       <-------------
 T3     SQL-------------->
 T4                       Trigger
 T5                       SQL
 T6                       CALL triggered------->
 T7                                             SP code
 T8                                     <-------SQL
 T9                                     ------->SP code
 T10                                    <-------SQL(UDF)
 T11                      Start UDF
 T12                      ------------------------------------>UDF code
 T13                      <------------------------------------SQL
 T14                      ------------------------------------>UDF code
 T16                      <------------------------------------UDF ends
 T17                      Back to SP    -------->SP code
 T18                      SQL           <--------Back to trigger
 T19                      Trigger ends
 T20   Code<--------------Return to Application
 T22   End
```

*Figure 127. Time spent executing nested activities*

Table 123 on page 880 shows the formula used to determine time for nested activities.

*Table 123. Sample for time used for execution of nested activities. TU = Time Used*

| Count for | Formula | Class |
|---|---|---|
| Application elapsed | T22-T1 | 1 |
| Application TCB (TU) | T22-T1 | 1 |
| Appl in DB2 elapsed | T2-T1 + T5-T3 + T20-T19 | 2 |
| Appl in DB2 TCB (TU) | T2-T1 + T5-T3 + T20-T19 | 2 |
| Trigger in DB2 elapsed | T6-T5 + T19-T18 | 2 |
| Trigger in DB2 TCB (TU) | T6-T5 + T19-T18 | 2 |
| Wait for STP time | T7-T6 | 3 |
| SP lapsed | T11-T6 + T18-T16 | 1 |
| SP TCB (TU) | T11-T6 + T18-T16 | 1 |
| SP SQL elapsed | T9-T8 + T11-T10 + T17-16 | 2 |
| SP SQL elapsed | T9-T8 + T11-T10 + T17-T16 | 2 |
| Wait for UDF time | T12-T11 | 3 |
| UDF elapsed | T16-T11 | 1 |
| UDF TCB (TU) | T16-T11 | 1 |
| UDF SQL elapsed | T14-T13 | 2 |
| UDF SQL TCB (TU) | T14-T13 | 2 |

The total class 2 time is the total of the ″in DB2″ times for the application, trigger, SP, and UDF. The class 3 ″wait″ times for the SPs and UDFs need to be added to the total class 3 times.

# Part 6. Appendixes

# Appendix A. DB2 sample tables

The information in this appendix is General-use Programming Interface and Associated Guidance Information as defined in “Notices” on page 1095.

Most of the examples in this book refer to the tables described in this appendix. As a group, the tables include information that describes employees, departments, projects, and activities, and make up a sample application that exemplifies most of the features of DB2. The sample storage group, databases, tablespaces, tables, and views are created when you run the installation sample jobs DSNTEJ1 and DSNTEJ7. DB2 sample objects that include LOBs are created in job DSNTEJ7. All other sample objects are created in job DSNTEJ1. The CREATE INDEX statements for the sample tables are not shown here; they, too, are created by the DSNTEJ1 and DSNTEJ7 sample jobs.

Authorization on all sample objects is given to PUBLIC in order to make the sample programs easier to run. The contents of any table can easily be reviewed by executing an SQL statement, for example SELECT * FROM DSN8710.PROJ. For convenience in interpreting the examples, the department and employee tables are listed here in full.

## Activity table (DSN8710.ACT)

The activity table describes the activities that can be performed during a project. The table resides in database DSN8D71A and is created with:

```
CREATE TABLE DSN8710.ACT
      (ACTNO    SMALLINT       NOT NULL,
       ACTKWD   CHAR(6)        NOT NULL,
       ACTDESC  VARCHAR(20)    NOT NULL,
       PRIMARY KEY (ACTNO)             )
  IN DSN8D71A.DSN8S71P
  CCSID EBCDIC;
```

## Content

Table 124 shows the content of the columns.

*Table 124. Columns of the activity table*

| Column | Column Name | Description |
|---|---|---|
| 1 | ACTNO | Activity ID (the primary key) |
| 2 | ACTKWD | Activity keyword (up to six characters) |
| 3 | ACTDESC | Activity description |

The activity table has these indexes:

*Table 125. Indexes of the activity table*

| Name | On Column | Type of Index |
|---|---|---|
| DSN8710.XACT1 | ACTNO | Primary, ascending |
| DSN8710.XACT2 | ACTKWD | Unique, ascending |

## Relationship to other tables

The activity table is a parent table of the project activity table, through a foreign key on column ACTNO.

# Department table (DSN8710.DEPT)

The department table describes each department in the enterprise and identifies its manager and the department to which it reports.

The table, shown in Table 128 on page 885, resides in table space DSN8D71A.DSN8S71D and is created with:

```
CREATE TABLE DSN8710.DEPT
      (DEPTNO    CHAR(3)           NOT NULL,
       DEPTNAME  VARCHAR(36)       NOT NULL,
       MGRNO     CHAR(6)                   ,
       ADMRDEPT  CHAR(3)           NOT NULL,
       LOCATION  CHAR(16)                  ,
       PRIMARY KEY (DEPTNO)                )
  IN DSN8D71A.DSN8S71D
  CCSID EBCDIC;
```

Because the table is self-referencing, and also is part of a cycle of dependencies, its foreign keys must be added later with these statements:

```
ALTER TABLE DSN8710.DEPT
      FOREIGN KEY RDD (ADMRDEPT) REFERENCES DSN8710.DEPT
              ON DELETE CASCADE;

ALTER TABLE DSN8710.DEPT
      FOREIGN KEY RDE (MGRNO) REFERENCES DSN8710.EMP
              ON DELETE SET NULL;
```

## Content

Table 126 shows the content of the columns.

*Table 126. Columns of the department table*

| Column | Column Name | Description |
|---|---|---|
| 1 | DEPTNO | Department ID, the primary key |
| 2 | DEPTNAME | A name describing the general activities of the department |
| 3 | MGRNO | Employee number (EMPNO) of the department manager |
| 4 | ADMRDEPT | ID of the department to which this department reports; the department at the highest level reports to itself |
| 5 | LOCATION | The remote location name |

The department table has these indexes:

*Table 127. Indexes of the department table*

| Name | On Column | Type of Index |
|---|---|---|
| DSN8710.XDEPT1 | DEPTNO | Primary, ascending |
| DSN8710.XDEPT2 | MGRNO | Ascending |
| DSN8710.XDEPT3 | ADMRDEPT | Ascending |

## Relationship to other tables

The table is self-referencing: the value of the administering department must be a department ID.

The table is a parent table of:
- The employee table, through a foreign key on column WORKDEPT
- The project table, through a foreign key on column DEPTNO.

It is a dependent of the employee table, through its foreign key on column MGRNO.

*Table 128. DSN8710.DEPT: department table*

| DEPTNO | DEPTNAME | MGRNO | ADMRDEPT | LOCATION |
|---|---|---|---|---|
| A00 | SPIFFY COMPUTER SERVICE DIV. | 000010 | A00 | ---------------- |
| B01 | PLANNING | 000020 | A00 | ---------------- |
| C01 | INFORMATION CENTER | 000030 | A00 | ---------------- |
| D01 | DEVELOPMENT CENTER | ------ | A00 | ---------------- |
| E01 | SUPPORT SERVICES | 000050 | A00 | ---------------- |
| D11 | MANUFACTURING SYSTEMS | 000060 | D01 | ---------------- |
| D21 | ADMINISTRATION SYSTEMS | 000070 | D01 | ---------------- |
| E11 | OPERATIONS | 000090 | E01 | ---------------- |
| E21 | SOFTWARE SUPPORT | 000100 | E01 | ---------------- |
| F22 | BRANCH OFFICE F2 | ------ | E01 | ---------------- |
| G22 | BRANCH OFFICE G2 | ------ | E01 | ---------------- |
| H22 | BRANCH OFFICE H2 | ------ | E01 | ---------------- |
| I22 | BRANCH OFFICE I2 | ------ | E01 | ---------------- |
| J22 | BRANCH OFFICE J2 | ------ | E01 | ---------------- |

The LOCATION column contains nulls until sample job DSNTEJ6 updates this column with the location name.

# Employee table (DSN8710.EMP)

The employee table identifies all employees by an employee number and lists basic personnel information.

The table shown in Table 131 on page 887 and Table 132 on page 888 resides in the partitioned table space DSN8D71A.DSN8S71E. Because it has a foreign key referencing DEPT, that table and the index on its primary key must be created first. Then EMP is created with:

```
CREATE TABLE DSN8710.EMP
      (EMPNO     CHAR(6)                          NOT NULL,
       FIRSTNME  VARCHAR(12)                      NOT NULL,
       MIDINIT   CHAR(1)                          NOT NULL,
       LASTNAME  VARCHAR(15)                      NOT NULL,
       WORKDEPT  CHAR(3)                                  ,
       PHONENO   CHAR(4)         CONSTRAINT NUMBER CHECK
                 (PHONENO >= '0000' AND
                  PHONENO <= '9999')                      ,
       HIREDATE  DATE                                     ,
       JOB       CHAR(8)                                  ,
       EDLEVEL   SMALLINT                                 ,
       SEX       CHAR(1)                                  ,
       BIRTHDATE DATE                                     ,
       SALARY    DECIMAL(9,2)                             ,
       BONUS     DECIMAL(9,2)                             ,
       COMM      DECIMAL(9,2)                             ,
       PRIMARY KEY (EMPNO)                                ,
       FOREIGN KEY RED (WORKDEPT) REFERENCES DSN8710.DEPT
                 ON DELETE SET NULL                      )
  EDITPROC  DSN8EAE1
  IN DSN8D71A.DSN8S71E
  CCSID EBCDIC;
```

## Content

Table 129 shows the content of the columns. The table has a check constraint, NUMBER, which checks that the phone number is in the numeric range 0000 to 9999.

*Table 129. Columns of the employee table*

| Column | Column Name | Description |
|---|---|---|
| 1 | EMPNO | Employee number (the primary key) |
| 2 | FIRSTNME | First name of employee |
| 3 | MIDINIT | Middle initial of employee |
| 4 | LASTNAME | Last name of employee |
| 5 | WORKDEPT | ID of department in which the employee works |
| 6 | PHONENO | Employee telephone number |
| 7 | HIREDATE | Date of hire |
| 8 | JOB | Job held by the employee |
| 9 | EDLEVEL | Number of years of formal education |
| 10 | SEX | Sex of the employee (M or F) |
| 11 | BIRTHDATE | Date of birth |
| 12 | SALARY | Yearly salary in dollars |
| 13 | BONUS | Yearly bonus in dollars |
| 14 | COMM | Yearly commission in dollars |

The table has these indexes:

*Table 130. Indexes of the employee table*

| Name | On Column | Type of Index |
|---|---|---|
| DSN8710.XEMP1 | EMPNO | Primary, partitioned, ascending |
| DSN8710.XEMP2 | WORKDEPT | Ascending |

## Relationship to other tables

The table is a parent table of:

- The department table, through a foreign key on column MGRNO
- The project table, through a foreign key on column RESPEMP.

It is a dependent of the department table, through its foreign key on column WORKDEPT.

*Table 131. Left half of DSN8710.EMP: employee table. Note that a blank in the MIDINIT column is an actual value of ' ' rather than null.*

| EMPNO | FIRSTNME | MIDINIT | LASTNAME | WORKDEPT | PHONENO | HIREDATE |
|---|---|---|---|---|---|---|
| 000010 | CHRISTINE | I | HAAS | A00 | 3978 | 1965-01-01 |
| 000020 | MICHAEL | L | THOMPSON | B01 | 3476 | 1973-10-10 |
| 000030 | SALLY | A | KWAN | C01 | 4738 | 1975-04-05 |
| 000050 | JOHN | B | GEYER | E01 | 6789 | 1949-08-17 |
| 000060 | IRVING | F | STERN | D11 | 6423 | 1973-09-14 |
| 000070 | EVA | D | PULASKI | D21 | 7831 | 1980-09-30 |
| 000090 | EILEEN | W | HENDERSON | E11 | 5498 | 1970-08-15 |
| 000100 | THEODORE | Q | SPENSER | E21 | 0972 | 1980-06-19 |
| 000110 | VINCENZO | G | LUCCHESSI | A00 | 3490 | 1958-05-16 |
| 000120 | SEAN |  | O’CONNELL | A00 | 2167 | 1963-12-05 |
| 000130 | DOLORES | M | QUINTANA | C01 | 4578 | 1971-07-28 |
| 000140 | HEATHER | A | NICHOLLS | C01 | 1793 | 1976-12-15 |
| 000150 | BRUCE |  | ADAMSON | D11 | 4510 | 1972-02-12 |
| 000160 | ELIZABETH | R | PIANKA | D11 | 3782 | 1977-10-11 |
| 000170 | MASATOSHI | J | YOSHIMURA | D11 | 2890 | 1978-09-15 |
| 000180 | MARILYN | S | SCOUTTEN | D11 | 1682 | 1973-07-07 |
| 000190 | JAMES | H | WALKER | D11 | 2986 | 1974-07-26 |
| 000200 | DAVID |  | BROWN | D11 | 4501 | 1966-03-03 |
| 000210 | WILLIAM | T | JONES | D11 | 0942 | 1979-04-11 |
| 000220 | JENNIFER | K | LUTZ | D11 | 0672 | 1968-08-29 |
| 000230 | JAMES | J | JEFFERSON | D21 | 2094 | 1966-11-21 |
| 000240 | SALVATORE | M | MARINO | D21 | 3780 | 1979-12-05 |
| 000250 | DANIEL | S | SMITH | D21 | 0961 | 1969-10-30 |
| 000260 | SYBIL | P | JOHNSON | D21 | 8953 | 1975-09-11 |
| 000270 | MARIA | L | PEREZ | D21 | 9001 | 1980-09-30 |
| 000280 | ETHEL | R | SCHNEIDER | E11 | 8997 | 1967-03-24 |
| 000290 | JOHN | R | PARKER | E11 | 4502 | 1980-05-30 |
| 000300 | PHILIP | X | SMITH | E11 | 2095 | 1972-06-19 |
| 000310 | MAUDE | F | SETRIGHT | E11 | 3332 | 1964-09-12 |
| 000320 | RAMLAL | V | MEHTA | E21 | 9990 | 1965-07-07 |
| 000330 | WING |  | LEE | E21 | 2103 | 1976-02-23 |
| 000340 | JASON | R | GOUNOT | E21 | 5698 | 1947-05-05 |
| 200010 | DIAN | J | HEMMINGER | A00 | 3978 | 1965-01-01 |
| 200120 | GREG |  | ORLANDO | A00 | 2167 | 1972-05-05 |
| 200140 | KIM | N | NATZ | C01 | 1793 | 1976-12-15 |
| 200170 | KIYOSHI |  | YAMAMOTO | D11 | 2890 | 1978-09-15 |
| 200220 | REBA | K | JOHN | D11 | 0672 | 1968-08-29 |
| 200240 | ROBERT | M | MONTEVERDE | D21 | 3780 | 1979-12-05 |
| 200280 | EILEEN | R | SCHWARTZ | E11 | 8997 | 1967-03-24 |
| 200310 | MICHELLE | F | SPRINGER | E11 | 3332 | 1964-09-12 |
| 200330 | HELENA |  | WONG | E21 | 2103 | 1976-02-23 |
| 200340 | ROY | R | ALONZO | E21 | 5698 | 1947-05-05 |

Table 132. Right half of DSN8710.EMP: employee table

| (EMPNO) | JOB | EDLEVEL | SEX | BIRTHDATE | SALARY | BONUS | COMM |
|---|---|---|---|---|---|---|---|
| (000010) | PRES | 18 | F | 1933-08-14 | 52750.00 | 1000.00 | 4220.00 |
| (000020) | MANAGER | 18 | M | 1948-02-02 | 41250.00 | 800.00 | 3300.00 |
| (000030) | MANAGER | 20 | F | 1941-05-11 | 38250.00 | 800.00 | 3060.00 |
| (000050) | MANAGER | 16 | M | 1925-09-15 | 40175.00 | 800.00 | 3214.00 |
| (000060) | MANAGER | 16 | M | 1945-07-07 | 32250.00 | 600.00 | 2580.00 |
| (000070) | MANAGER | 16 | F | 1953-05-26 | 36170.00 | 700.00 | 2893.00 |
| (000090) | MANAGER | 16 | F | 1941-05-15 | 29750.00 | 600.00 | 2380.00 |
| (000100) | MANAGER | 14 | M | 1956-12-18 | 26150.00 | 500.00 | 2092.00 |
| (000110) | SALESREP | 19 | M | 1929-11-05 | 46500.00 | 900.00 | 3720.00 |
| (000120) | CLERK | 14 | M | 1942-10-18 | 29250.00 | 600.00 | 2340.00 |
| (000130) | ANALYST | 16 | F | 1925-09-15 | 23800.00 | 500.00 | 1904.00 |
| (000140) | ANALYST | 18 | F | 1946-01-19 | 28420.00 | 600.00 | 2274.00 |
| (000150) | DESIGNER | 16 | M | 1947-05-17 | 25280.00 | 500.00 | 2022.00 |
| (000160) | DESIGNER | 17 | F | 1955-04-12 | 22250.00 | 400.00 | 1780.00 |
| (000170) | DESIGNER | 16 | M | 1951-01-05 | 24680.00 | 500.00 | 1974.00 |
| (000180) | DESIGNER | 17 | F | 1949-02-21 | 21340.00 | 500.00 | 1707.00 |
| (000190) | DESIGNER | 16 | M | 1952-06-25 | 20450.00 | 400.00 | 1636.00 |
| (000200) | DESIGNER | 16 | M | 1941-05-29 | 27740.00 | 600.00 | 2217.00 |
| (000210) | DESIGNER | 17 | M | 1953-02-23 | 18270.00 | 400.00 | 1462.00 |
| (000220) | DESIGNER | 18 | F | 1948-03-19 | 29840.00 | 600.00 | 2387.00 |
| (000230) | CLERK | 14 | M | 1935-05-30 | 22180.00 | 400.00 | 1774.00 |
| (000240) | CLERK | 17 | M | 1954-03-31 | 28760.00 | 600.00 | 2301.00 |
| (000250) | CLERK | 15 | M | 1939-11-12 | 19180.00 | 400.00 | 1534.00 |
| (000260) | CLERK | 16 | F | 1936-10-05 | 17250.00 | 300.00 | 1380.00 |
| (000270) | CLERK | 15 | F | 1953-05-26 | 27380.00 | 500.00 | 2190.00 |
| (000280) | OPERATOR | 17 | F | 1936-03-28 | 26250.00 | 500.00 | 2100.00 |
| (000290) | OPERATOR | 12 | M | 1946-07-09 | 15340.00 | 300.00 | 1227.00 |
| (000300) | OPERATOR | 14 | M | 1936-10-27 | 17750.00 | 400.00 | 1420.00 |
| (000310) | OPERATOR | 12 | F | 1931-04-21 | 15900.00 | 300.00 | 1272.00 |
| (000320) | FIELDREP | 16 | M | 1932-08-11 | 19950.00 | 400.00 | 1596.00 |
| (000330) | FIELDREP | 14 | M | 1941-07-18 | 25370.00 | 500.00 | 2030.00 |
| (000340) | FIELDREP | 16 | M | 1926-05-17 | 23840.00 | 500.00 | 1907.00 |
| (200010) | SALESREP | 18 | F | 1933-08-14 | 46500.00 | 1000.00 | 4220.00 |
| (200120) | CLERK | 14 | M | 1942-10-18 | 29250.00 | 600.00 | 2340.00 |
| (200140) | ANALYST | 18 | F | 1946-01-19 | 28420.00 | 600.00 | 2274.00 |
| (200170) | DESIGNER | 16 | M | 1951-01-05 | 24680.00 | 500.00 | 1974.00 |
| (200220) | DESIGNER | 18 | F | 1948-03-19 | 29840.00 | 600.00 | 2387.00 |
| (200240) | CLERK | 17 | M | 1954-03-31 | 28760.00 | 600.00 | 2301.00 |
| (200280) | OPERATOR | 17 | F | 1936-03-28 | 26250.00 | 500.00 | 2100.00 |
| (200310) | OPERATOR | 12 | F | 1931-04-21 | 15900.00 | 300.00 | 1272.00 |
| (200330) | FIELDREP | 14 | F | 1941-07-18 | 25370.00 | 500.00 | 2030.00 |
| (200340) | FIELDREP | 16 | M | 1926-05-17 | 23840.00 | 500.00 | 1907.00 |

## Employee photo and resume table (DSN8710.EMP_PHOTO_RESUME)

The employee photo and resume table complements the employee table. Each row of the photo and resume table contains a photo of the employee, in two formats, and the employee's resume. The photo and resume table resides in table space DSN8D71A.DSN8S71E. The following statement creates the table:

```
CREATE TABLE  DSN8710.EMP_PHOTO_RESUME
               (EMPNO      CHAR(06) NOT NULL,
                EMP_ROWID  ROWID NOT NULL GENERATED ALWAYS,
                PSEG_PHOTO BLOB(100K),
                BMP_PHOTO  BLOB(100K),
                RESUME        CLOB(5K))
                PRIMARY KEY  EMPNO
        IN  DSN8D71L.DSN8S71B
        CCSID  EBCDIC;
```

DB2 requires an auxiliary table for each LOB column in a table. These statements define the auxiliary tables for the three LOB columns in DSN8710.EMP_PHOTO_RESUME:

```
CREATE AUX TABLE DSN8710.AUX_BMP_PHOTO
               IN DSN8D71L.DSN8S71M
               STORES DSN8710.EMP_PHOTO_RESUME
               COLUMN BMP_PHOTO;

CREATE AUX TABLE DSN8710.AUX_PSEG_PHOTO
               IN DSN8D71L.DSN8S71L
               STORES DSN8710.EMP_PHOTO_RESUME
               COLUMN PSEG_PHOTO;

CREATE AUX TABLE DSN8710.AUX_EMP_RESUME
               IN DSN8D71L.DSN8S71N
               STORES DSN8710.EMP_PHOTO_RESUME
               COLUMN RESUME;
```

## Content

Table 133 shows the content of the columns.

*Table 133. Columns of the employee photo and resume table*

| Column | Column Name | Description |
|---|---|---|
| 1 | EMPNO | Employee ID (the primary key) |
| 2 | EMP_ROWID | Row ID to uniquely identify each row of the table. DB2 supplies the values of this column. |
| 3 | PSEG_PHOTO | Employee photo, in PSEG format |
| 4 | BMP_PHOTO | Employee photo, in BMP format |
| 5 | RESUME | Employee resume |

The employee photo and resume table has these indexes:

*Table 134. Indexes of the employee photo and resume table*

| Name | On Column | Type of Index |
|---|---|---|
| DSN8710.XEMP_PHOTO_RESUME | EMPNO | Primary, ascending |

The auxiliary tables for the employee photo and resume table have these indexes:

*Table 135. Indexes of the auxiliary tables for the employee photo and resume table*

| Name | On Table | Type of Index |
|---|---|---|
| DSN8710.XAUX_BMP_PHOTO | DSN8710.AUX_BMP_PHOTO | Unique |
| DSN8710.XAUX_PSEG_PHOTO | DSN8710.AUX_PSEG_PHOTO | Unique |
| DSN8710.XAUX_EMP_RESUME | DSN8710.AUX_EMP_RESUME | Unique |

## Relationship to other tables

The table is a parent table of the project table, through a foreign key on column RESPEMP.

# Project table (DSN8710.PROJ)

The project table describes each project that the business is currently undertaking. Data contained in each row include the project number, name, person responsible, and schedule dates.

The table resides in database DSN8D71A. Because it has foreign keys referencing DEPT and EMP, those tables and the indexes on their primary keys must be created first. Then PROJ is created with:

```
CREATE TABLE DSN8710.PROJ
               (PROJNO   CHAR(6) PRIMARY KEY NOT NULL,
                PROJNAME VARCHAR(24)    NOT NULL WITH DEFAULT
                  'PROJECT NAME UNDEFINED',
                DEPTNO   CHAR(3)        NOT NULL REFERENCES
                  DSN8710.DEPT ON DELETE RESTRICT,
                RESPEMP  CHAR(6)        NOT NULL REFERENCES
                  DSN8710.EMP ON DELETE RESTRICT,
                PRSTAFF  DECIMAL(5, 2)          ,
                PRSTDATE DATE                   ,
                PRENDATE DATE                   ,
                MAJPROJ  CHAR(6))
        IN DSN8D71A.DSN8S71P
        CCSID EBCDIC;
```

Because the table is self-referencing, the foreign key for that restraint must be added later with:

```
ALTER TABLE DSN8710.PROJ
      FOREIGN KEY RPP (MAJPROJ) REFERENCES DSN8710.PROJ
              ON DELETE CASCADE;
```

# Content

Table 136 shows the content of the columns.

*Table 136. Columns of the project table*

| Column | Column Name | Description |
|---|---|---|
| 1 | PROJNO | Project ID (the primary key) |
| 2 | PROJNAME | Project name |
| 3 | DEPTNO | ID of department responsible for the project |
| 4 | RESPEMP | ID of employee responsible for the project |
| 5 | PRSTAFF | Estimated mean number of persons needed between PRSTDATE and PRENDATE to achieve the whole project, including any subprojects |
| 6 | PRSTDATE | Estimated project start date |
| 7 | PRENDATE | Estimated project end date |
| 8 | MAJPROJ | ID of any project of which this project is a part |

The project table has these indexes:

*Table 137. Indexes of the project table*

| Name | On Column | Type of Index |
|---|---|---|
| DSN8710.XPROJ1 | PROJNO | Primary, ascending |
| DSN8710.XPROJ2 | RESPEMP | Ascending |

## Relationship to other tables

The table is self-referencing: a nonnull value of MAJPROJ must be a project number. The table is a parent table of the project activity table, through a foreign key on column PROJNO. It is a dependent of:

- The department table, through its foreign key on DEPTNO
- The employee table, through its foreign key on RESPEMP.

# Project activity table (DSN8710.PROJACT)

The project activity table lists the activities performed for each project. The table resides in database DSN8D71A. Because it has foreign keys referencing PROJ and ACT, those tables and the indexes on their primary keys must be created first. Then PROJACT is created with:

```
CREATE TABLE DSN8710.PROJACT
      (PROJNO    CHAR(6)                       NOT NULL,
       ACTNO     SMALLINT                      NOT NULL,
       ACSTAFF   DECIMAL(5,2)                          ,
       ACSTDATE  DATE                          NOT NULL,
       ACENDATE  DATE                                  ,
       PRIMARY KEY (PROJNO, ACTNO, ACSTDATE),
       FOREIGN KEY RPAP (PROJNO) REFERENCES DSN8710.PROJ
                                 ON DELETE RESTRICT,
       FOREIGN KEY RPAA (ACTNO) REFERENCES DSN8710.ACT
                                 ON DELETE RESTRICT)
  IN DSN8D71A.DSN8S71P
  CCSID EBCDIC;
```

## Content

Table 138 shows the content of the columns.

*Table 138. Columns of the project activity table*

| Column | Column Name | Description |
|---|---|---|
| 1 | PROJNO | Project ID |
| 2 | ACTNO | Activity ID |
| 3 | ACSTAFF | Estimated mean number of employees needed to staff the activity |
| 4 | ACSTDATE | Estimated activity start date |
| 5 | ACENDATE | Estimated activity completion date |

The project activity table has this index:

*Table 139. Index of the project activity table*

| Name | On Columns | Type of Index |
|---|---|---|
| DSN8710.XPROJAC1 | PROJNO, ACTNO, ACSTDATE | primary, ascending |

## Relationship to other tables

The table is a parent table of the employee to project activity table, through a foreign key on columns PROJNO, ACTNO, and EMSTDATE. It is a dependent of:

- The activity table, through its foreign key on column ACTNO
- The project table, through its foreign key on column PROJNO

## Employee to project activity table (DSN8710.EMPPROJACT)

The employee to project activity table identifies the employee who performs an activity for a project, tells the proportion of the employee's time required, and gives a schedule for the activity.

The table resides in database DSN8D71A. Because it has foreign keys referencing EMP and PROJACT, those tables and the indexes on their primary keys must be created first. Then EMPPROJACT is created with:

```
CREATE TABLE DSN8710.EMPPROJACT
      (EMPNO     CHAR(6)                        NOT NULL,
       PROJNO    CHAR(6)                        NOT NULL,
       ACTNO     SMALLINT                       NOT NULL,
       EMPTIME   DECIMAL(5,2)                           ,
       EMSTDATE  DATE                                   ,
       EMENDATE  DATE                                   ,
       FOREIGN KEY REPAPA (PROJNO, ACTNO, EMSTDATE)
                 REFERENCES DSN8710.PROJACT
                                      ON DELETE RESTRICT,
       FOREIGN KEY REPAE (EMPNO) REFERENCES DSN8710.EMP
                                      ON DELETE RESTRICT)
  IN DSN8D71A.DSN8S71P
  CCSID EBCDIC;
```

## Content

Table 140 shows the content of the columns.

*Table 140. Columns of the employee to project activity table*

| Column | Column Name | Description |
|---|---|---|
| 1 | EMPNO | Employee ID number |
| 2 | PROJNO | Project ID of the project |
| 3 | ACTNO | ID of the activity within the project |
| 4 | EMPTIME | A proportion of the employee's full time (between 0.00 and 1.00) to be spent on the activity |
| 5 | EMSTDATE | Date the activity starts |
| 6 | EMENDATE | Date the activity ends |

The table has these indexes:

*Table 141. Indexes of the employee to project activity table*

| Name | On Columns | Type of Index |
|---|---|---|
| DSN8710.XEMPPROJACT1 | PROJNO, ACTNO, EMSTDATE, EMPNO | Unique, ascending |
| DSN8710.XEMPPROJACT2 | EMPNO | Ascending |

## Relationship to other tables

The table is a dependent of:

- The employee table, through its foreign key on column EMPNO
- The project activity table, through its foreign key on columns PROJNO, ACTNO, and EMSTDATE.

## Relationships among the tables

Figure 128 shows relationships among the tables. These are established by foreign keys in dependent tables that reference primary keys in parent tables. You can find descriptions of the columns with descriptions of the tables.

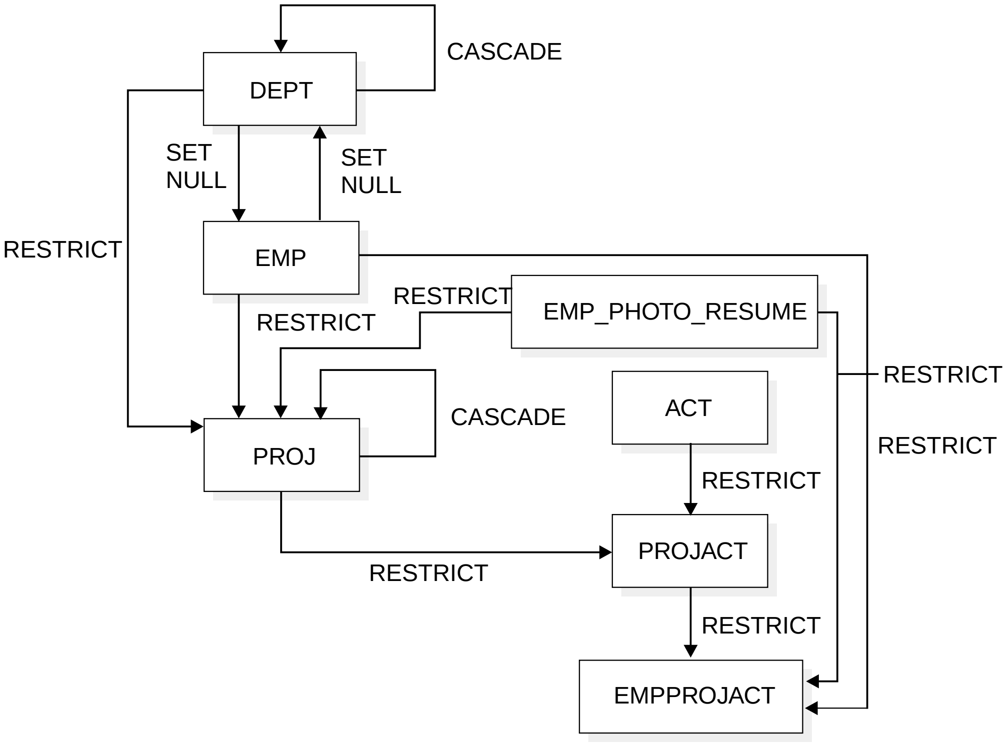


*Figure 128. Relationships among tables in the sample application. Arrows point from parent tables to dependent tables.*

## Views on the sample tables

DB2 creates a number of views on the sample tables for use in the sample applications. Table 142 indicates the tables on which each view is defined and the sample applications that use the view. All view names have the qualifier DSN8710.

*Table 142. Views on sample tables*

| View name | On tables or views | Used in application |
|---|---|---|
| VDEPT | DEPT | Organization<br>Project |
| VHDEPT | DEPT | Distributed organization |
| VEMP | EMP | Distributed organization<br>Organization<br>Project |
| VPROJ | PROJ | Project |
| VACT | ACT | Project |
| VEMPPROJACT | EMPROJACT | Project |
| VDEPMG1 | DEPT<br>EMP | Organization |

*Table 142. Views on sample tables (continued)*

| View name | On tables or views | Used in application |
|---|---|---|
| VEMPDPT1 | DEPT<br>EMP | Organization |
| VASTRDE1 | DEPT | |
| VASTRDE2 | VDEPMG1<br>EMP | Organization |
| VPROJRE1 | PROJ<br>EMP | Project |
| VPSTRDE1 | VPROJRE1<br>VPROJRE2 | Project |
| VPSTRDE2 | VPROJRE1 | Project |
| VSTAFAC1 | PROJACT<br>ACT | Project |
| VSTAFAC2 | EMPPROJACT<br>ACT<br>EMP | Project |
| VPHONE | EMP<br>DEPT | Phone |
| VEMPLP | EMP | Phone |

The SQL statements that create the sample views are shown below.

```
CREATE VIEW DSN8710.VDEPT
   AS SELECT ALL      DEPTNO  ,
                      DEPTNAME,
                      MGRNO   ,
                      ADMRDEPT
   FROM DSN8710.DEPT;
```

```
 CREATE VIEW DSN8710.VHDEPT
    AS SELECT ALL      DEPTNO  ,
                       DEPTNAME,
                       MGRNO   ,
                       ADMRDEPT,
                       LOCATION
    FROM DSN8710.DEPT;
```

```
CREATE VIEW DSN8710.VEMP
   AS SELECT ALL      EMPNO   ,
                      FIRSTNME,
                      MIDINIT ,
                      LASTNAME,
                      WORKDEPT
   FROM DSN8710.EMP;
```

```
CREATE VIEW DSN8710.VPROJ
   AS SELECT ALL
             PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF,
             PRSTDATE, PRENDATE, MAJPROJ
   FROM DSN8710.PROJ ;
```

```
CREATE VIEW DSN8710.VACT
      AS SELECT ALL   ACTNO   ,
                      ACTKWD  ,
                      ACTDESC
           FROM DSN8710.ACT ;
```

```
CREATE VIEW DSN8710.VPROJACT
      AS SELECT ALL
            PROJNO,ACTNO, ACSTAFF, ACSTDATE, ACENDATE
            FROM DSN8710.PROJACT ;
```

```
CREATE VIEW DSN8710.VEMPPROJACT
      AS SELECT ALL
          EMPNO, PROJNO, ACTNO, EMPTIME, EMSTDATE, EMENDATE
          FROM DSN8710.EMPPROJACT ;

CREATE VIEW DSN8710.VDEPMG1
      (DEPTNO, DEPTNAME, MGRNO, FIRSTNME, MIDINIT,
       LASTNAME, ADMRDEPT)
    AS SELECT ALL
       DEPTNO, DEPTNAME, EMPNO, FIRSTNME, MIDINIT,
        LASTNAME, ADMRDEPT
        FROM DSN8710.DEPT LEFT OUTER JOIN DSN8710.EMP
        ON MGRNO = EMPNO ;

CREATE VIEW DSN8710.VEMPDPT1
      (DEPTNO, DEPTNAME, EMPNO, FRSTINIT, MIDINIT,
       LASTNAME, WORKDEPT)
    AS SELECT ALL
       DEPTNO, DEPTNAME, EMPNO, SUBSTR(FIRSTNME, 1, 1), MIDINIT,
       LASTNAME, WORKDEPT
        FROM DSN8710.DEPT RIGHT OUTER JOIN DSN8710.EMP
        ON WORKDEPT = DEPTNO ;

CREATE VIEW DSN8710.VASTRDE1
    (DEPT1NO,DEPT1NAM,EMP1NO,EMP1FN,EMP1MI,EMP1LN,TYPE2,
     DEPT2NO,DEPT2NAM,EMP2NO,EMP2FN,EMP2MI,EMP2LN)
    AS SELECT ALL
        D1.DEPTNO,D1.DEPTNAME,D1.MGRNO,D1.FIRSTNME,D1.MIDINIT,
        D1.LASTNAME, '1',
        D2.DEPTNO,D2.DEPTNAME,D2.MGRNO,D2.FIRSTNME,D2.MIDINIT,
        D2.LASTNAME
        FROM DSN8710.VDEPMG1 D1, DSN8710.VDEPMG1 D2
        WHERE D1.DEPTNO = D2.ADMRDEPT ;

CREATE VIEW DSN8710.VASTRDE2
    (DEPT1NO,DEPT1NAM,EMP1NO,EMP1FN,EMP1MI,EMP1LN,TYPE2,
     DEPT2NO,DEPT2NAM,EMP2NO,EMP2FN,EMP2MI,EMP2LN)
    AS SELECT ALL
         D1.DEPTNO,D1.DEPTNAME,D1.MGRNO,D1.FIRSTNME,D1.MIDINIT,
         D1.LASTNAME,'2',
         D1.DEPTNO,D1.DEPTNAME,E2.EMPNO,E2.FIRSTNME,E2.MIDINIT,
         E2.LASTNAME
         FROM DSN8710.VDEPMG1 D1, DSN8710.EMP E2
         WHERE D1.DEPTNO = E2.WORKDEPT;

CREATE VIEW DSN8710.VPROJRE1
  (PROJNO,PROJNAME,PROJDEP,RESPEMP,FIRSTNME,MIDINIT,
   LASTNAME,MAJPROJ)
   AS SELECT ALL
      PROJNO,PROJNAME,DEPTNO,EMPNO,FIRSTNME,MIDINIT,
      LASTNAME,MAJPROJ
     FROM DSN8710.PROJ, DSN8710.EMP
     WHERE RESPEMP = EMPNO ;

CREATE VIEW DSN8710.VPSTRDE1
  (PROJ1NO,PROJ1NAME,RESP1NO,RESP1FN,RESP1MI,RESP1LN,
   PROJ2NO,PROJ2NAME,RESP2NO,RESP2FN,RESP2MI,RESP2LN)
   AS SELECT ALL
        P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
        P1.LASTNAME,
        P2.PROJNO,P2.PROJNAME,P2.RESPEMP,P2.FIRSTNME,P2.MIDINIT,
        P2.LASTNAME
     FROM DSN8710.VPROJRE1 P1,
       DSN8710.VPROJRE1 P2
     WHERE P1.PROJNO = P2.MAJPROJ ;

CREATE VIEW DSN8710.VPSTRDE2
  (PROJ1NO,PROJ1NAME,RESP1NO,RESP1FN,RESP1MI,RESP1LN,
   PROJ2NO,PROJ2NAME,RESP2NO,RESP2FN,RESP2MI,RESP2LN)
   AS SELECT ALL
        P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
        P1.LASTNAME,
        P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
        P1.LASTNAME
     FROM DSN8710.VPROJRE1 P1
       WHERE NOT EXISTS
         (SELECT * FROM DSN8710.VPROJRE1 P2
           WHERE P1.PROJNO = P2.MAJPROJ) ;

CREATE VIEW DSN8710.VFORPLA
  (PROJNO,PROJNAME,RESPEMP,PROJDEP,FRSTINIT,MIDINIT,LASTNAME)
   AS SELECT ALL
      F1.PROJNO,PROJNAME,RESPEMP,PROJDEP, SUBSTR(FIRSTNME, 1, 1),
      MIDINIT, LASTNAME
     FROM DSN8710.VPROJRE1 F1 LEFT OUTER JOIN DSN8710.EMPPROJACT F2
     ON F1.PROJNO = F2.PROJNO;

CREATE VIEW DSN8710.VSTAFAC1
  (PROJNO, ACTNO, ACTDESC, EMPNO, FIRSTNME, MIDINIT, LASTNAME,
   EMPTIME,STDATE,ENDATE, TYPE)
   AS SELECT ALL
         PA.PROJNO, PA.ACTNO, AC.ACTDESC,' ', ' ', ' ', ' ',
         PA.ACSTAFF, PA.ACSTDATE,
         PA.ACENDATE,'1'
     FROM DSN8710.PROJACT PA, DSN8710.ACT AC
     WHERE PA.ACTNO = AC.ACTNO ;

CREATE VIEW DSN8710.VSTAFAC2
  (PROJNO, ACTNO, ACTDESC, EMPNO, FIRSTNME, MIDINIT, LASTNAME,
   EMPTIME,STDATE, ENDATE, TYPE)
   AS SELECT ALL
         EP.PROJNO, EP.ACTNO, AC.ACTDESC, EP.EMPNO,EM.FIRSTNME,
         EM.MIDINIT, EM.LASTNAME, EP.EMPTIME, EP.EMSTDATE,
         EP.EMENDATE,'2'
     FROM DSN8710.EMPPROJACT EP, DSN8710.ACT AC, DSN8710.EMP EM
     WHERE EP.ACTNO = AC.ACTNO  AND EP.EMPNO = EM.EMPNO ;

CREATE VIEW DSN8710.VPHONE
             (LASTNAME,
              FIRSTNAME,
              MIDDLEINITIAL,
              PHONENUMBER,
              EMPLOYEENUMBER,
              DEPTNUMBER,
              DEPTNAME)
   AS SELECT ALL      LASTNAME,
                      FIRSTNME,
                      MIDINIT ,
                      VALUE(PHONENO,'    '),
                      EMPNO,
                      DEPTNO,
                      DEPTNAME
   FROM DSN8710.EMP, DSN8710.DEPT
   WHERE WORKDEPT = DEPTNO;

CREATE VIEW DSN8710.VEMPLP
             (EMPLOYEENUMBER,
              PHONENUMBER)
   AS SELECT ALL      EMPNO   ,
                      PHONENO
   FROM DSN8710.EMP ;
```

## Storage of sample application tables

Figure 129 on page 897 shows how the sample tables are related to databases and storage groups. Two databases are used to illustrate the possibility. Normally, related data is stored in the same database.

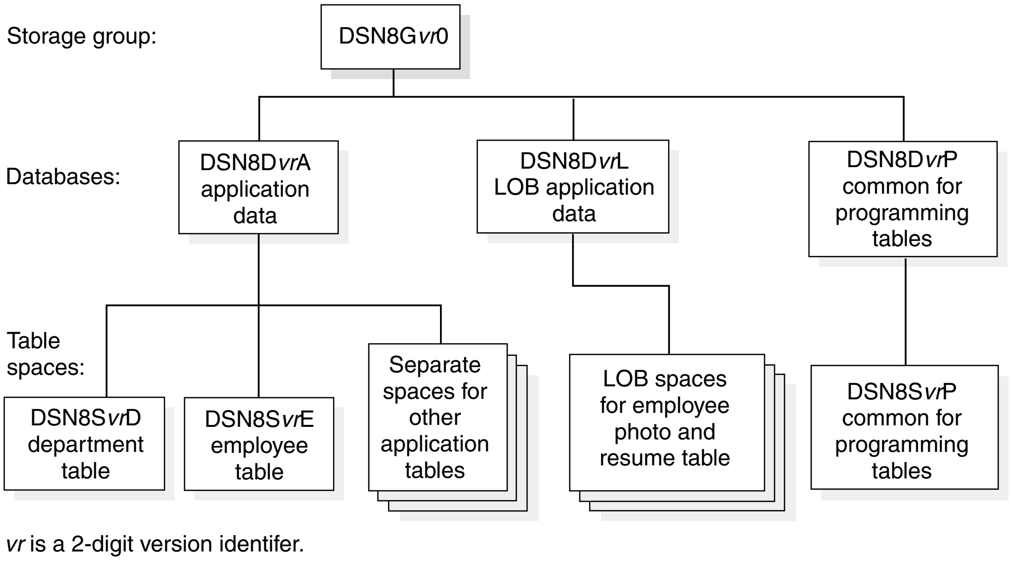

Storage group:
DSN8Gvr0
Databases:
DSN8DvrA application data
DSN8DvrL LOB application data
DSN8DvrP common for programming tables
Table spaces:
DSN8SvrD department table
DSN8SvrE employee table
Separate spaces for other application tables
LOB spaces for employee photo and resume table
DSN8SvrP common for programming tables
vr is a 2-digit version identifer.

*Figure 129. Relationship among sample databases and table spaces*

In addition to the storage group and databases shown in Figure 129, the storage group DSN8G71U and database DSN8D71U are created when you run DSNTEJ2A.

## Storage group

The default storage group, SYSDEFLT, created when DB2 is installed, is not used to store sample application data. The storage group used to store sample application data is defined by this statement:

```
CREATE STOGROUP DSN8G710
  VOLUMES (DSNV01)
  VCAT  DSNC710;
```

## Databases

The default database, created when DB2 is installed, is not used to store the sample application data. Two databases are used: one for tables related to applications, the other for tables related to programs. They are defined by the following statements:

```
CREATE DATABASE DSN8D71A
  STOGROUP DSN8G710
  BUFFERPOOL BP0
  CCSID EBCDIC;

CREATE DATABASE DSN8D71P
  STOGROUP DSN8G710
  BUFFERPOOL BP0
  CCSID EBCDIC;

CREATE DATABASE DSN8D71L
  STOGROUP DSN8G710
  BUFFERPOOL BP0
  CCSID EBCDIC;
```

## Table spaces

The following table spaces are explicitly defined by the statements shown below. The table spaces not explicitly defined are created implicitly in the DSN8D71A database, using the default space attributes.

```
CREATE TABLESPACE DSN8S71D
  IN DSN8D71A
  USING STOGROUP DSN8G710
        PRIQTY 20
        SECQTY 20
        ERASE NO
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;

CREATE TABLESPACE DSN8S71E
  IN DSN8D71A
  USING STOGROUP DSN8G710
        PRIQTY 20
        SECQTY 20
        ERASE NO
  NUMPARTS 4
    (PART 1 USING STOGROUP DSN8G710
                  PRIQTY 12
                  SECQTY 12,
     PART 3 USING STOGROUP DSN8G710
                  PRIQTY 12
                  SECQTY 12)
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  COMPRESS YES
  CCSID EBCDIC;
CREATE TABLESPACE DSN8S71B
  IN DSN8D71L
  USING STOGROUP DSN8G710
        PRIQTY 20
        SECQTY 20
        ERASE NO
  LOCKSIZE PAGE
  LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;
CREATE LOB TABLESPACE DSN8S71M
  IN DSN8D71L
  LOG NO;

CREATE LOB TABLESPACE DSN8S71L
  IN DSN8D71L
  LOG NO;

CREATE LOB TABLESPACE DSN8S71N
  IN DSN8D71L
  LOG NO;
```

```
CREATE TABLESPACE DSN8S71C
  IN DSN8D71P
  USING STOGROUP DSN8G710
        PRIQTY 160
        SECQTY 80
  SEGSIZE 4
  LOCKSIZE TABLE
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;

CREATE TABLESPACE DSN8S71P
  IN DSN8D71A
  USING STOGROUP DSN8G710
        PRIQTY 160
        SECQTY 80
  SEGSIZE 4
  LOCKSIZE ROW
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;
CREATE TABLESPACE DSN8S71R
  IN DSN8D71A
  USING STOGROUP DSN8G710
        PRIQTY 20
        SECQTY 20
        ERASE NO
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;
CREATE TABLESPACE DSN8S71S
   IN DSN8D71A
   USING STOGROUP DSN8G710
             PRIQTY 20
             SECQTY 20
             ERASE NO
   LOCKSIZE PAGE LOCKMAX SYSTEM
   BUFFERPOOL BP0
   CLOSE NO
   CCSID EBCDIC;
```

# Appendix B. Writing exit routines

The information in this appendix is Product-sensitive Programming Interface and Associated Guidance Information as defined in “Notices” on page 1095.

DB2 provides installation-wide exit points to routines that you provide. They are described under the following headings:

- “Connection and sign-on routines”
- “Access control authorization exit” on page 909
- “Edit routines” on page 921
- “Validation routines” on page 925
- “Date and time routines” on page 927
- “Conversion procedures” on page 931
- “Field procedures” on page 934
- “Log capture routines” on page 944
- “Routines for dynamic plan selection in CICS” on page 946

## Connection and sign-on routines

Your DB2 subsystem has two exit points for authorization routines, one in connection processing and one in sign-on processing. They perform crucial steps in the assignment of values to the primary, secondary, and SQL IDs. You must have a routine for each exit. Default routines are provided for both—DSN3@ATH for connections and DSN3@SGN for sign-ons.

If your installation has a connection exit routine and you are planning to use CONNECT with the USER/USING clause, you should examine your exit routine and take the following into consideration. The security-related control blocks that are normally associated with the thread or the address space that your exit can access will not be updated by DB2 to reflect the userid and password specified in the USER/USING clause of the CONNECT statement.

For a general view of the roles of the exit routines in assigning authorization IDs, see “Chapter 12. Controlling access to a DB2 subsystem” on page 169. That description can show that you can most easily provide the security features you want by assigning identifiers through RACF or some similar program and using the sample connection and sign-on routines provided by IBM. This section describes the interfaces for those routines and the functions they provide. If you want to have secondary authorization IDs, you must replace the default routines with the sample routines or with routines of your own.

### General considerations

“General considerations for writing exit routines” on page 950 applies to these routines. One exception to the description of execution environments is that the routines execute in non-cross-memory mode.

### Specifying the routines

Your connection routine must have a CSECT name and entry point of DSN3@ATH. Its load module name can be the same, but need not be. Your sign-on routine must have a CSECT name and entry point of DSN3@SGN. Its load module name can be the same, but need not be.

You can use an ALIAS statement of the linkage editor to provide the entry point name.

Default routines with those names and entry points already exist in library *prefix*.SDSNLOAD; to use your routines instead, place them in library *prefix*.SDSNEXIT. You can use the install job DSNTIJEX to assemble and link-edit the routines and place them in the new library. If you use any other library, you could have to change the STEPLIB or JOBLIB concatenations in the DB2 start-up procedures.

You can combine both routines into one CSECT and load module if you wish, but the module must include both entry points, DSN3@ATH and DSN3@SGN. Use standard assembler and linkage editor control statements to define the entry points. DB2 loads the module twice at startup, by issuing the MVS LOAD macro first for entry point DSN3@ATH and then for entry point DSN3@SGN. However, because the routines are reentrant, only one copy of each remains in virtual storage.

## Sample exit routines

The sample exit routines provide examples of the functions and interfaces described below. They are provided in source code as members of *prefix*.SDSNSAMP. To examine the sample connection routine, list or assemble member DSN3SATH; for the sample sign-on routine, member DSN3SSGN. To assemble, you must use Assembler H; both routines use features not available in Assembler XF.

***Change required for some CICS users:*** If you attach to DB2 with an AUTH parameter in the RCT other than AUTH=GROUP, you also have the RACF list-of-groups option active, and you have transactions whose initial primary authorization ID is not defined to RACF, then you must change the sample sign-on exit routine (DSN3SSGN) before assembling and using it. Proceed as follows:

1. In the source code, locate this statement:

   ```
   SSGN035   DS     0H         BLANK BACKSCAN LOOP REENTRY
   ```
2. Nearby, locate this statement:

   ```
             B      SSGN037    ENTIRE NAME IS BLANK, LEAVE
   ```

   (At this writing, its line number is 03664000, but that is subject to change.)
3. Replace the previous statement with this one:

   ```
             B      SSGN090    NO GROUP NAME...  BYPASS RACF CHECK
   ```

The change avoids an abend with SQLCODE -922 in the situation described above. With the change, DB2 does not use RACF group names unless you use AUTH=GROUP; for other values of AUTH, the routine provides no secondary IDs.

## When exits are taken

Different local processes enter the access control procedure at different points, depending on the environment they originate from. (Quite different criteria apply to remote requests; they are described in “Controlling requests from remote applications” on page 176.)

- These processes go through connection processing only:
  - Requests originating in TSO foreground and background (including online utilities and requests through the call attachment facility)
  - JES-initiated batch jobs
  - Requests through started task control address spaces (from the MVS START command)

- These processes go through connection processing and can later go through the sign-on exit also.
  – The IMS control region
  – The CICS recovery coordination task
  – DL/I batch
  – Requests through the Recoverable Resource Manager Services attachment facility (RRSAF)
- These processes go through sign-on processing:
  – Requests from IMS dependent regions (including MPP, BMP, and Fast Path)
  – CICS transaction subtasks

For instructions on controlling the IDs associated with connection requests, see “Processing connections” on page 170. For instructions on controlling the IDs associated with sign-on requests, see “Processing sign-ons” on page 173.

## EXPL for connection and sign-on routines

Figure 130 shows how the parameter list points to other information.

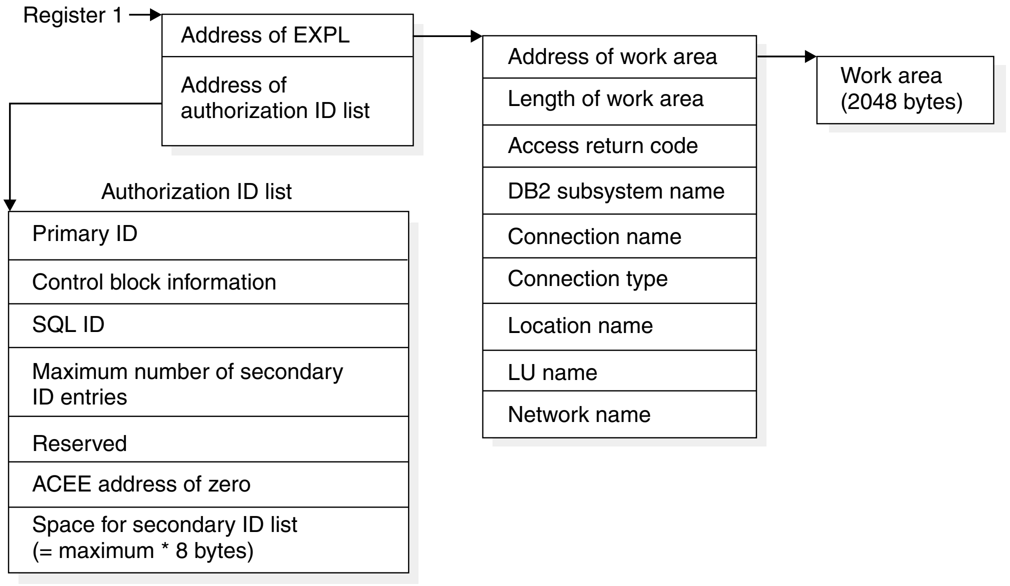


*Figure 130. How a connection or sign-on parameter list points to other information*

## Exit parameter list

Connection and sign-on routines use 28 bytes more of the exit parameter list EXPL than do other routines. The table that follows shows the entire list. The exit parameter list is described by macro DSNDEXPL.

*Table 143. Exit parameter list for connection and sign-on routines*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| EXPLWA | 0 | Address | Address of a 2048-byte work area to be used by the routine |
| EXPLWL | 4 | Signed 4-byte integer | Length of the work area, in bytes; value is 2048. |
| EXPLRSV1 | 8 | Signed 2-byte integer | Reserved |

*Table 143. Exit parameter list for connection and sign-on routines (continued)*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| EXPLRC1 | A | Signed 2-byte integer | Not used |
| EXPLRC2 | C | Signed 4-byte integer | Not used |
| EXPLARC | 10 | Signed 4-byte integer | Access return code. Values can be:<br>**0** Access allowed; DB2 continues processing.<br>**12** Access denied; DB2 terminates processing with an error. |
| EXPLSSNM | 14 | Character, 8 bytes | DB2 subsystem name, left justified; for example, 'DSN ' |
| EXPLCONN | 1C | Character, 8 bytes | Connection name for requesting location |
| EXPLTYPE | 24 | Character, 8 bytes | Connection type for requesting location. For DDF threads, the connection type is 'DIST '. |
| EXPLSITE | 2C | Character, 16 bytes | For SNA protocols, this is the location name of the requesting location or <*luname*>. For TCP/IP protocols, this is the dotted decimal IP address of the requester. |
| EXPLLUNM | 3C | Character, 8 bytes | For SNA protocols, this is the locally known LU name of the requesting location. For TCP/IP protocols, this is the character string 'TCPIP '. |
| EXPLNTID | 44 | Character, 17 bytes | For SNA protocols, the fully qualified network name of the requesting location. For TCP/IP protocols, this field is reserved. |

## Authorization ID parameter list

The second parameter list, which is specific to connection and sign-on routines, is called an authorization ID list. Its contents are shown in Table 144. The description is given by macro DSNDAIDL.

*Table 144. Authorization ID list for a connection or sign-on exit routine*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| AIDLPRIM | 0 | Character, 8 bytes | Primary authorization ID for input and output; see descriptions in the text. |
| AIDLCODE | 8 | Character, 2 bytes | Control block identifier |
| AIDLTLEN | A | Signed 2-byte integer | Total length of control block |
| AIDLEYE | C | Character, 4 bytes | Eyecatcher for block, “AIDL” |
| AIDLSQL | 10 | Character, 8 bytes | On output, the current SQL ID |
| AIDLSCNT | 18 | Signed 4-byte integer | Number of entries allocated to secondary authorization ID list. Always equal to 245. |

*Table 144. Authorization ID list for a connection or sign-on exit routine (continued)*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| AIDLSAPM | 1C | Address | For a sign-on routine only, the address of an 8-character additional authorization ID. If RACF is active, the ID is the user ID's connected group name. If the address was not provided, the field contains zero. |
| AIDLCKEY | 20 | Character, 1 byte | Storage key of the ID pointed to by AIDLSAPM. To move that ID, use the “move with key” (MVCK) instruction, specifying this key. |
| AIDLRSV1 | 21 | Character, 3 bytes | Reserved |
| AIDLRSV2 | 24 | Signed 4-byte integer | Reserved |
| AIDLACEE | 28 | Signed 4-byte integer | The address of the ACEE structure, if known; otherwise, zero |
| AIDLRACL | 2C | Signed 4-byte integer | Length of data area returned by RACF, plus 4 bytes |
| AIDLRACR | 30 | 26 bytes | Reserved |
| AIDLSEC | 4A | Character, maximum x 8 bytes | List of the secondary authorization IDs, 8 bytes each |

## Input values

The primary authorization ID has been placed first in the authorization ID list for compatibility with DB2 Version 1. The default routines, and any authorization routine you might have written for DB2 Version 1, accept only the first item for input.

The input values of the several authorization IDs are as follows:

### For a connection routine

1. The initial primary authorization ID for a local request can be obtained from the MVS address space extension block (ASXB).

   The ASXB contains at most only a 7-character value. That is always sufficient for a TSO user ID or a user ID from an MVS JOB statement, and the ASXB is always used for those cases.

   For CICS, IMS, or other started tasks, MVS can also pass an 8-character ID. If an 8-character ID is available, and if its first 7 characters agree with the ASXB value, then DB2 uses the 8-character ID. Otherwise it uses the ASXB value.

   You can alter the sample exit routine to use the ASXB value always. For instructions, see “Processing in the sample routines” on page 907.

   If RACF is active, the field used contains a verified RACF user ID; otherwise, it contains blanks.
2. The primary ID for a remote request is the ID passed in the conversation attach request header (SNA FMH5) or in the DRDA SECCHK command.
3. The SQL ID contains blanks.
4. The list of secondary IDs contains blanks.

### For a sign-on routine

1. The initial primary ID is determined. See “Processing sign-ons” on page 173 for information about how the primary ID is determined.
2. The SQL ID and all secondary IDs contain blanks.
3. Field AIDLSAPM in the authorization ID list can contain the address of an 8-character additional authorization ID, obtained by the CICS attachment facility using the RACROUTE REQUEST=EXTRACT service with the requester's user ID. If RACF is active, this ID is the RACF-connected group name from the ACEE corresponding to the requester's user ID. Otherwise, this field is blanks. IMS Version 2 Release 2 does not pass this parameter.
4. Field AIDLCKEY contains the storage key of the identifier pointed to by AIDLSAPM. To move that ID, use the “move with key” (MVCK) instruction, specifying this key.
5. Field AIDLACEE contains the ACEE address only for a sign-on through the CICS attachment facility and only when the CICS RCT uses AUTH=GROUP.

## Expected output

DB2 uses the output values of the primary, SQL, and secondary IDs. Your routines can set those to any value that is an SQL short identifier. If your identifier does not meet the 8-character criteria, the request is abended. Pad shorter identifiers on the right with blanks. If the values returned are not blank, DB2 interprets them as follows:

1. The primary ID becomes the primary authorization ID.
2. The list of secondary IDs, down to the first blank entry or to a maximum of 245 entries, becomes the list of secondary authorization IDs. The space allocated for the secondary ID list is only large enough to contain the maximum number of authorization IDs. This number is in field AIDLSCNT and is currently 245. If you do not restrict the number of secondary authorization IDs to 245, disastrous results (like abends and storage overlays) can occur.
3. The SQL ID is checked to see if it is the same as the primary or one of the secondary IDs. If it is not, the connection or sign-on process abends. Otherwise, the validated ID becomes the current SQL ID.

If the returned value of the primary ID *is* blank, DB2 takes the following steps:

- In connection processing, the default ID defined when DB2 was installed (UNKNOWN AUTHID on panel DSNTIPP) is substituted as the primary authorization ID and the current SQL ID. The list of secondary IDs is set to blanks.
- Sign-on processing abends; there is no default value of the primary ID.

If the returned value of the SQL ID is blank, DB2 makes it equal to the value of the primary ID. If the list of secondary IDs is blank, it is left so; there are no default secondary IDs.

Your routine must also set a return code in word 5 of the exit parameter list to allow or deny access (field EXPLARC). By those means you can deny the connection altogether. The code must have one of the following values; any other value causes abends:

| Value | Meaning |
|---|---|
| **0** | Access allowed; continue processing |
| **12** | Access denied; terminate |

## Processing in the sample routines

The sample routines provided by IBM can serve as models of the processing required in connection and sign-on routines. To write a routine that implements your own choices, it can be easiest to modify the samples. Both routines have similar sections for setup, constants, and storage areas. Both routines set values of the primary, SQL, and secondary IDs in three numbered sections, which perform the following functions:

***In the sample connection routine (DSN3SATH):***

Section 1 provides the same function as in the default connection routine. It tests whether the first character of the input primary ID is greater than blank.

- If the first character is greater, the value is not changed.
- If the first character is not greater, the value is set to:
  - The logon ID, if the request is from a TSO foreground address space.
  - The job user ID from the JES job control table.

  If, after the above processing is done, no primary ID has been located, Section 2 is bypassed.

At the beginning of Section 2, you can restore one commented-out instruction that then truncates the primary authorization ID to 7 characters. (The instruction is identified by comments in the code.) Section 2 next tests RACF options and makes the following changes in the list of secondary IDs, which is initially blank:

- If RACF is not active, leave the list blank.
- If the list of groups option is not active, but there is an ACEE, copy the connected group name as the only secondary ID.
- If the list of groups option is active, copy the list of group names from the ICHPCGRP block into AIDLSEC in the authorization ID list.

Section 3 takes the following steps:

1. Make the SQL ID equal to the primary ID.

   If a TSO data set name prefix cannot be found, bypass the remainder of Section 3.
2. If the TSO data set name prefix is a valid primary or secondary ID, replace the SQL ID with the TSO data set name prefix. Otherwise, leave the default (primary ID) as the SQL ID.

***In the sample sign-on routine (DSN3SSGN):***

Section 1 leaves the primary ID alone.

Section 2 sets the SQL ID to the value of the primary ID.

Section 3 tests RACF options and makes the following changes in the list of secondary IDs, which is initially blank:

- If RACF is not active, leave the list blank.
- If the list of groups option is active, attempt to find an existing ACEE from which to copy the authorization ID list.
  - If AIDLACEE contains a value other than zero, validate that it is an ACEE and use it.

    Otherwise, look for a valid ACEE chained from the TCB or from the ASXB or, if no usable ACEE exists, issue RACROUTE to have RACF build an ACEE structure for the primary ID.

    Copy the list of group names from the ACEE structure into the secondary authorization list.
  - If the exit issued RACROUTE to build an ACEE, issue another RACROUTE macro to have the structure deleted.

– If a list of secondary authorization IDs has not been built, and AIDLSAPM is not zero, copy the data pointed to by AIDLSAPM into AIDLSEC.

## Performance considerations

Your sign-on exit routine is part of the critical path for transaction processing in IMS or CICS, so you want it to execute as quickly as possible. Avoid writing SVC calls like GETMAIN, FREEMAIN, and ATTACH, or I/O operations to any data set or database. You might want to delete the list of groups processing in Section 3 of the sample sign-on exit.

The sample sign-on exit routine can issue the RACF RACROUTE macro with the default option SMC=YES. If another product issues RACROUTE with SMC=NO, a deadlock could occur. The situation has been of concern in the CICS environment and might occur in IMS.

Your routine can also possibly enhance the performance of later authorization checking. Authorization for dynamic SQL statements is checked first for the CURRENT SQLID, then the primary authorization ID, and then the secondary authorization IDs. If you know that a user's privilege most often comes from a secondary authorization ID, then set the CURRENT SQLID to this secondary ID within your exit routine.

## Debugging your exit routine

The diagnostic aids described below can assist in debugging connection and sign-on exit routines.

***Subsystem support identify recovery:*** The identify ESTAE recovery routine, DSN3IDES, generates the following VRADATA entries. The last entry, key VRAIMO, is generated only if the abend occurred within the connection exit routine.

| VRA keyname | Key hex value | Data length | Content |
|---|---|---|---|
| VRAFPI | 22 | 8 | Constant 'IDESTRAK' |
| VRAFP | 23 | 24 | 32-bit recovery tracking flags. 32-bit integer AGNT block unique identifier. AGNT block address. AIDL block address. Initial primary authorization ID as copied from ASXBUSER. |
| VRAIMO | 7C | 10 | Connection exit load module load point address. Connection exit entry point address. Offset of failing address in the PSW from the connection exit entry point address. |

***Subsystem support sign-on recovery:*** The sign-on ESTAE recovery routine DSN3SIES generates the following VRADATA entries. The last entry, key VRAIMO, is generated only if the abend occurred within the sign-on exit routine.

| VRA keyname | Key hex value | Data length | Content |
|---|---|---|---|
| VRAFPI | 22 | 8 | Constant 'SIESTRAK' |
| VRAFP | 23 | 20 | Primary authorization ID (CCBUSER). AGNT block address. Identify-level CCB block address. Sign-on-level CCB block address |

| VRA keyname | Key hex value | Data length | Content |
|---|---|---|---|
| VRAIMO | 7C | 10 | Sign-on exit load module load point address. Sign-on exit entry point address. Offset of failing address in the PSW from the sign-on exit entry point address. |

***Diagnostics for connection and sign-on exits:*** The connection (identify) and sign-on recovery routines provide diagnostics for the corresponding exit routines. The diagnostics are produced only when the abend occurred in the exit routine.

- Dump Title:

  The component failing module name is "DSN3@ATH" for a connection exit or "DSN3@SGN" for a sign-on exit.
- MVS and RETAIN® symptom data:

  SDWA symptom data fields SDWACSCT (CSECT/) and SDWAMODN (MOD/) are set to "DSN3@ATH" or "DSN3@SGN", as appropriate.

  The component subfunction code (SUB1/ or VALU/C) is set to "SSSC#DSN3@ATH#IDENTIFY" or "SSSC#DSN3@SGN#SIGNON", as appropriate.
- Summary Dump Additions.

  The AIDL, if addressable, and the SADL, if present, are included in the summary dump for the failing allied agent. If the failure occurred in connection or sign-on processing, the exit parameter list (EXPL) is also included. If the failure occurred in the system services address space, the entire SADL storage pool is included in the summary dump.

## Access control authorization exit

DB2 provides an exit point that lets you provide your own access control authorization exit routine, or lets RACF (Security Server for OS/390 Release 4, or subsequent releases), or an equivalent security system perform DB2 authorization checking for SQL and utilities. Your routine specifies whether the authorization checking should all be done by RACF, or partly by RACF and DB2. (Also, the routine can be called and still let all checking be performed by DB2.) For more information about how to use the routine provided by the Security Server, see *OS/390 Security Server (RACF) Security Administrator's Guide*.

When DB2 invokes the routine, it passes three possible functions to the routine:

- Initialization (DB2 startup)
- Authorization check
- Termination (DB2 shutdown)

The bulk of the work in the routine is for authorization checking. When DB2 must determine the authorization for a privilege, it invokes your routine. The routine determines the authorization for the privilege and then indicates to DB2 whether authorized, not authorized, or whether DB2 should do its own authorization check, instead.

***When the exit routine is bypassed***: In the following situations, the exit routine is not called to check authorization:

- The user has installation SYSADM or installation SYSOPR authority. This authorization check is made strictly within DB2.

- DB2 security has been disabled (NO on the USE PROTECTION field of installation panel DSNTIPP).
- Authorization has been cached from a prior check.
- From a prior invocation of the exit routine, the routine had indicated that it should not be called again.
- GRANT statements.

## General considerations

The routine executes in the *ssnm*DBM1 address space of DB2.

"General considerations for writing exit routines" on page 950 applies to this routine, but with the following exceptions to the description of execution environments:

- The routine executes in non-cross-memory mode during initialization and termination (XAPLFUNC of 1 or 3, described in Table 145 on page 913).
- During authorization checking the routine can execute under a TCB or SRB in cross-memory or non-cross-memory mode.

## Specifying the routine

Your access control authorization routine must have a CSECT name and entry point of DSNX@XAC. The load module name or alias name must also be DSNX@XAC. A default routine with this name and entry point exists in library *prefix*.SDSNLOAD; to use your routine instead, place it in library *prefix*.SDSNEXIT. Use installation job DSNTIJEX to assemble and link-edit the routine and to place it in the new APF-authorized library. If you use any other library, you might have to change the STEPLIB or JOBLIB concatenations in the DB2 start-up procedures.

The source code for the default routine is in *prefix*.SDSNSAMP as DSNXSXAC. You can use it to write your own exit routine. To assemble it, you must use Assembler H.

## The default routine

The default exit routine returns a code to the DB2 authorization module indicating that a user-defined access control authorization exit routine is not available. DB2 then performs normal authorization checking and does not attempt to invoke this exit again.

## When the exit is taken

This exit is taken in three instances:

- At DB2 startup.

  When DB2 starts, this exit is taken to allow the external authorization checking application to perform any required setup prior to authorization checking. An example of a required setup task is loading authorization profiles into storage. DB2 uses the reason code that the exit routine sets during startup to determine how to handle exception situations. See "Exception processing" on page 920 for details.
- When an authorization check is to be performed on a privilege.

  At the point when DB2 would access security tables in the catalog, to check authorization on a privilege, this exit is taken. This exit is only taken if none of the prior invocations have indicated that the exit must not be called again.
- At DB2 shutdown.

When DB2 is stopping, this exit is taken to let the external authorization checking application perform its cleanup before DB2 stops.

## Other considerations for using the access control authorization exit

Here are some other things to be aware of when you use an access control authorization exit routine:

- Plan for what to do if DB2 cannot provide an ACEE

  Sometimes DB2 cannot provide an ACEE. For example, if you are not using external security in CICS (that is, SEC=NO is specified in the DFHSIT), CICS does not pass an ACEE to the CICS attachment facility. When DB2 does not have an ACEE, it passes zeros in the XAPLACEE field. If this happens, your routine can return a 4 in the EXPLRC1 field, and let DB2 handle the authorization check.

  DB2 does not pass the ACEE address for DB2 commands or IMS transactions. The ACEE address is passed for CICS transactions, if available.

- Authorization ID, ACEE relationship

  XAPL has two authorization ID fields, XAPLUPRM (the primary authorization ID) and XAPLUCHK (the authorization ID that DB2 uses to perform the authorization). These two fields might have different values.

  The ACEE passed in XAPLACEE is that of the primary authorization ID, XAPLUPRM.

- Invalid or inoperative plans and packages

  In DB2, when a privilege required by a plan or package is revoked, the plan or package is invalidated. If you use an authorization access control routine, it cannot tell DB2 that a privilege is revoked. Therefore, DB2 cannot know to invalidate the plan or package.

  If the revoked privilege was EXECUTE on a user-defined function, DB2 marks the plan or package inoperative instead of invalid.

  If a privilege that the plan or package depends on is revoked, and if you want to invalidate the plan or package or make it inoperative, you must use the SQL GRANT statement to grant the revoked privilege and then use the SQL REVOKE statement to revoke it.

- Dropping views

  In DB2, when a privilege required to create a view is revoked the view is dropped. Similar to the revocation of plan privileges, such an event is not communicated to DB2 by the authorization checking routine.

  If you want DB2 to drop the view when a privilege is revoked, you must use the SQL statements GRANT and REVOKE.

- Caching of EXECUTE on plans

  The results of authorization checks on the EXECUTE privilege are not cached when those checks are performed by the exit routine.

- Caching of EXECUTE on packages and routines

  The results of authorization checks on the EXECUTE privilege for packages and routines are cached (assuming that package and routine authorization caching is enabled on your system). If this privilege is revoked in the exit routine, the cached information is not updated to reflect the revoke. You must use the SQL GRANT and REVOKE statements to update the cached information.

- Caching of dynamic SQL statements

  Dynamic statements can be cached when they have passed the authorization checks (assuming that dynamic statement caching is enabled on your system). If the privileges that this statement requires are revoked from the authorization ID that is cached with the statement, then this cached statement must be invalidated. If the privilege is revoked in the exit routine this does not happen, and you must use the SQL statements GRANT and REVOKE to refresh the cache.

- Resolution of user-defined functions

  The create timestamp for the user-defined function must be older than the bind timestamp for the package or plan in which the user-defined function is invoked. If DB2 authorization checking is in effect, and DB2 performs an automatic rebind on a plan or package that invokes a user-defined function, any user-defined functions that were created after the original BIND or REBIND of the invoking plan or package are not candidates for execution.

  If you use an access control authorization exit routine, some user-defined functions that were not candidates for execution before the original BIND or REBIND of the invoking plan or package might become candidates for execution during the automatic rebind of the invoking plan or package. If a user-defined function is invoked during an automatic rebind, and that user-defined function is invoked from a trigger body and receives a transition table, the form of the invoked function that DB2 uses for function selection includes only the columns of the transition table that existed at the time of the original BIND or REBIND of the package or plan for the invoking program.

## Parameter list for the access control authorization routine

Figure 131 shows how the parameter list points to other information.

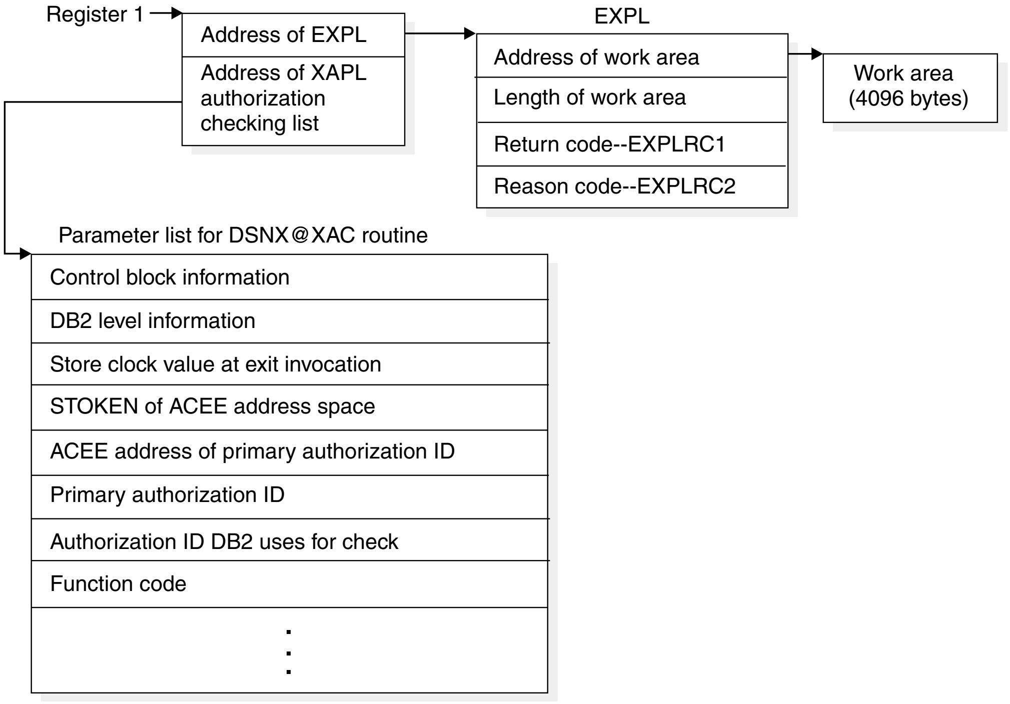

Register 1
Address of EXPL
Address of XAPL authorization checking list
EXPL
Address of work area
Length of work area
Return code--EXPLRC1
Reason code--EXPLRC2
Work area (4096 bytes)
Parameter list for DSNX@XAC routine
Control block information
DB2 level information
Store clock value at exit invocation
STOKEN of ACEE address space
ACEE address of primary authorization ID
Primary authorization ID
Authorization ID DB2 uses for check
Function code
⋮

*Figure 131. How an authorization routine's parameter list points to other information*

The work area (4096 bytes) is obtained once during the startup of DB2 and only released when DB2 is shut down. The work area is shared by all invocations to the exit routine.

## Exit parameter list (XAPL)

At invocation, registers are set as described in “Registers at invocation” on page 951, and the authorization checking routine uses the standard exit parameter list (EXPL) described there. Table 145 shows the exit-specific parameter list, described by macro DSNDXAPL.

*Table 145. Parameter list for the access control authorization routine. Field names indicated by an asterisk (*) apply to initialization, termination, and authorization checking. Other fields apply to authorization checking only.*

| Name | Hex offset | Data type | Input or output | Description |
|---|---|---|---|---|
| XAPLCBID* | 0 | Character, 2-byte integer | Input | Control block identifier; value X'216A'. |
| XAPLLEN * | 2 | Signed, 2-byte integer | Input | Length of XAPL; value X'100' (decimal 256). |
| XAPLEYE * | 4 | Character, 4 bytes | Input | Control block eye catcher; value “XAPL”. |
| XAPLLVL * | 8 | Character, 8 bytes | Input | DB2 version and level; for example, “VxRxMx    ”. |
| XAPLSTCK * | 10 | Character, 8 bytes | Input | The store clock value when the exit is invoked. Use this to correlate information to this specific invocation. |
| XAPLSTKN * | 18 | Character, 8 bytes | Input | STOKEN of the address space in which XAPLACEE resides. Binary zeroes indicate that XAPLACEE is in the home address space. |
| XAPLACEE * | 20 | Address | Input | ACEE address:<br>• Of the DB2 address space (*ssnm*DBM1) when XAPLFUNC is 1 or 3.<br>• Of the primary authorization ID associated with this agent when XAPLFUNC is 2.<br>There may be cases were an ACEE address is not available for an agent. In such cases this field contains zero. |
| XAPLUPRM * | 24 | Character, 8 bytes | Input | One of the following IDs:<br>• When XAPLFUNC is 1 or 3, it contains the User ID of the DB2 address space (*ssnm*DBM1)<br>• When XAPLFUNC is 2, it contains the primary authorization ID associated with the agent |
| XAPLUCHK | 2C | Character, 8 bytes | Input | Authorization ID on which DB2 performs the check. It could be the primary, secondary, or some other ID. |
| XAPLFUNC * | 34 | Signed, 2-byte integer | Input | Function to be performed by exit routine<br>**1** Initialization<br>**2** Authorization Check<br>**3** Termination |
| XAPLGPAT * | 36 | Character, 4 bytes | Input | DB2 group attachment name for data sharing. The DB2 subsystem name if not data sharing. |
| XAPLRSV1 | 3A | Character, 4 bytes | | Reserved |

*Table 145. Parameter list for the access control authorization routine (continued). Field names indicated by an asterisk (*) apply to initialization, termination, and authorization checking. Other fields apply to authorization checking only.*

| Name | Hex offset | Data type | Input or output | Description |
|---|---|---|---|---|
| XAPLTYPE | 3E | Character,1 | Input | DB2 object type:<br>**D** Database<br>**R** Table space<br>**T** Table<br>**P** Application plan<br>**K** Package<br>**S** Storage group<br>**C** Collection<br>**B** Buffer pool<br>**U** System privilege<br>**E** Distinct type<br>**F** User-defined function<br>**M** Schema<br>**O** Stored procedure<br>**J** JAR |
| XAPLFLG1 | 3F | Character,1 | Input | The highest-order bit, bit 8, (XAPLCHKS) is on if the secondary IDs associated with this authorization ID (XAPLUCHK) are included in DB2's authorization check. If it is off, only this authorization ID is checked.<br>The next highest-order bit, bit 7, (XAPLUTB) is on if this is a table privilege (SELECT, INSERT, and so on) and if SYSCTRL is not sufficient authority to perform the specified operation on a table. SYSCTRL does not have the privilege of accessing user data unless specifically granted to it.<br>The next highest-order bit, bit 6, (XAPLAUTO) is on if this is an AUTOBIND. See “Access control authorization exit” on page 909 for more information on function resolution during an AUTOBIND.<br>The next highest-order bit, bit 5, (XAPLCRVW) is on if the installation parameter DBADM CREATE AUTH is set to YES.<br>The remaining 4 bits are reserved. |
| XAPLOBJN | 40 | Character, 20 bytes | Input | Unqualified name of the object with which the privilege is associated. It is one of the following names:<br>Name Length<br>**Database** 8<br>**Table space** 8<br>**Table** 18<br>**Application plan** 8<br>**Package** 8 |

*Table 145. Parameter list for the access control authorization routine (continued). Field names indicated by an asterisk (*) apply to initialization, termination, and authorization checking. Other fields apply to authorization checking only.*

| Name | Hex offset | Data type | Input or output | Description |
|---|---|---|---|---|
| | | | | **Storage group** 8<br>**Collection** 18<br>**Buffer pool** 8<br>**Schema** 8<br>**Distinct type** 18<br>**User-defined function** 18<br>**JAR** 18<br>For special system privileges (SYSADM, SYSCTRL, and so on) this field might be blank. See macro DSNXAPRV.<br>This parameter is left-justified and padded with blanks. If not applicable, it contains blanks or binary zeros. |
| XAPLOWNQ | 54 | Character, 20 bytes | Input | Object owner (creator) or object qualifier. The contents of this parameter depends on either the privilege being checked or the object. See Table 147 on page 917.<br>This parameter is left-justified and padded with blanks. If not applicable, it contains blanks or binary zeros. |
| XAPLREL1 | 68 | Character, 20 bytes | Input | Other related information. The contents of this parameter depends on either the privilege being checked or the object. See Table 147 on page 917.<br>This parameter is left-justified and padded with blanks. If not applicable, it contains blanks or binary zeros. |
| XAPLREL2 | 7C | Character, 64 bytes | Input | Other related information. The contents of this parameter depends on the privilege being checked. See Table 147 on page 917.<br>This parameter is left-justified and padded with blanks. If not applicable, it contains blanks or binary zeros. |
| XAPLPRIV | BC | Signed, 2-byte integer | Input | DB2 privilege being checked. See macro DSNXAPRV for a complete list of privileges. |
| XAPLFROM | BE | Character, 1 byte | Input | Source of the request:<br>**S** Remote request that uses DB2 private protocol.<br>**’ ’** Not a remote request that uses DB2 private protocol.<br>DB2 authorization restricts remote requests that use DB2 private protocol to the SELECT, UPDATE, INSERT and DELETE privileges. |
| XAPLXBTS | BF | Timestamp, 10 bytes | Input | The function resolution timestamp. Authorizations received prior to this timestamp are valid.<br>Applicable to functions and procedures. See *DB2 SQL Reference* for more information on function resolution. |
| XAPLRSV2 | C9 | Character, 5 bytes | | Reserved |

*Table 145. Parameter list for the access control authorization routine (continued). Field names indicated by an asterisk (*) apply to initialization, termination, and authorization checking. Other fields apply to authorization checking only.*

| Name | Hex offset | Data type | Input or output | Description |
|---|---|---|---|---|
| XAPLONWT | CE | Character, 1 byte | Output | Information required by DB2 from the exit routine for the UPDATE and REFERENCES table privileges:<br>Value — Explanation<br>**' '** — Requester has privilege on the entire table<br>***** — Requester has privilege on just this column<br>See macro DSNXAPRV for definition of these privileges. |
| XAPLDIAG | CF | Character, 40 bytes | Output | Information returned by the exit routine to help diagnose problems. |
| XAPLRSV3 | F7 | Character, 9 bytes | | Reserved |

Table 146 has database information for determining authorization for creating a view. The address to this parameter list is in XAPLREL2. See Table 147 on page 917 for more information on CREATE VIEW.

*Table 146. Parameter list for the access control authorization routine—database information*

| Name | Hex offset | Data type | Input or output | Description |
|---|---|---|---|---|
| XAPLDBNP | 0 | Address | Input | Address of information for the next database. X'00000000' indicates no next database exists. |
| XAPLDBNM | 4 | Character, 8 bytes | Input | Database name. |

*Table 146. Parameter list for the access control authorization routine—database information (continued)*

| Name | Hex offset | Data type | Input or output | Description |
|---|---|---|---|---|
| XAPLDBDA | C | Character, 1 byte | Output | Required by DB2 from the exit routine for CREATE VIEW.<br>A value of Y indicates the user ID in field XAPLUCHK has database administrator authority on the database in field XAPLDBNM.<br>When the exit checks if XAPLUCHK can create a view for another authorization ID, it first checks for SYSADM or SYSCTRL authority. If the check is successful, no more checking is necessary because SYSADM or SYSCTRL authority satisfies the requirement that the view owner has the SELECT privilege for all tables and views that the view may be based on.<br>If the authorization ID does not have SYSADM or SYSCTRL authority, the exit checks if the view creator has DBADM on each database of the tables that the view is based on because the DBADM authority on the database of the base table satisfies the requirement that the view owner has the SELECT privilege for all base tables in that database. |
| XAPLRSV5 | D | Character, 3 bytes | none | Reserved |

XAPLOWNQ, XAPLREL1 and XAPLREL2 might further qualify the object or may provide additional information that can be used in determining authorization for certain privileges. These privileges and the contents of XAPLOWNQ, XAPLREL1 and XAPLREL2 are shown in Table 147.

*Table 147. Related information for certain privileges*

| Privilege | Object type (XAPLTYPE) | XAPLOWNQ | XAPLREL1 | XAPLREL2 |
|---|---|---|---|---|
| 0053 (UPDATE)<br>0054 (REFERENCES) | T | Table Name Qualifier | Column Name if applicable | Database name |

*Table 147. Related information for certain privileges (continued)*

| Privilege | Object type (XAPLTYPE) | XAPLOWNQ | XAPLREL1 | XAPLREL2 |
|---|---|---|---|---|
| 0022 (CATMAINT CONVERT)<br>0050 (SELECT)<br>0051 (INSERT)<br>0052 (DELETE)<br>0055 (TRIGGER)<br>0056 (CREATE INDEX)<br>0061 (ALTER)<br>0073 (DROP)<br>0075 (LOAD)<br>0076 (CHANGE NAME QUALIFIER)<br>0097 (COMMENT ON)<br>0098 (LOCK)<br>0102 (CREATE SYNONYM)<br>0233 (ANY TABLE PRIVILEGE) | T | Table name qualifier | blank | Database name |
| 0020 (DROP ALIAS)<br>0104 (DROP SYNONYM) | T | Object name qualifier | blank | blank |
| 0103 (ALTER INDEX)<br>0105 (DROP INDEX)<br>0274 (COMMENT ON INDEX) | T | Object name qualifier | blank | Database name |
| 0108 (CREATE VIEW) | T | blank | blank | First 4 bytes has the address to Database Information. Blanks indicate no database information has been passed. |
| 0065 (BIND) | P | Plan owner | blank | blank |
| 0064 (EXECUTE) | K | Collection ID | blank | blank |
| 0065 (BIND) | K | Collection ID | Package owner | blank |
| 0073 (DROP) | K | Collection ID | blank | Version ID |
| 0225 (COPY ON PKG) | K | Collection ID | Package owner | blank |
| 0228 (ALLPKAUT) | K | Collection ID | blank | blank |
| 0229 (SUBPKAUT) | K | Collection ID | blank | blank |
| 0061 (ALTER) | R | Database name | blank | blank |
| 0073 (DROP) | R | Database name | blank | blank |
| 0087 (USE) | R | Database name | blank | blank |
| 0227 (BIND AGENT) | U | Package owner | blank | blank |
| 0015 (CREATE ALIAS) | U | blank | blank | Database name |
| 0263 (USAGE) | E | Schema name | Distinct type owner | blank |
| 0263 (USAGE) | J | Schema name | JAR owner | blank |
| 0064 (EXECUTE)<br>0265 (START)<br>0266 (STOP)<br>0267 (DISPLAY) | F | Schema name | User-defined function owner | blank |

*Table 147. Related information for certain privileges (continued)*

| Privilege | Object type (XAPLTYPE) | XAPLOWNQ | XAPLREL1 | XAPLREL2 |
|---|---|---|---|---|
| 0064 (EXECUTE)<br>0265 (START)<br>0266 (STOP)<br>0267 (DISPLAY) | O | Schema name | Procedure owner | blank |
| 0252 (ALTERIN)<br>0097 (COMMENT ON)<br>0252 (DROPIN) | M | Schema name | Object owner | blank |

The data types and field lengths of the information shown in Table 147 on page 917 is shown in Table 148.

*Table 148. Data types and field lengths*

| Resource name or other | Type | Length |
|---|---|---|
| Database name | Character | 8 |
| Table name qualifier | Character | 8 |
| Object name qualifier | Character | 8 |
| Column name | Character | 18 |
| Collection ID | Character | 18 |
| Plan owner | Character | 8 |
| Package owner | Character | 8 |
| Package version ID | Character | 64 |
| Schema name | Character | 8 |
| Distinct type owner | Character | 8 |
| JAR owner | Character | 8 |
| User-defined function owner | Character | 8 |
| Procedure owner | Character | 8 |

## Expected output

Your authorization exit routine is expected to return certain fields when it is called. These output fields are indicated in Table 145 on page 913. If an unexpected value is returned in any of these fields an abend occurs. Register 3 points to the field in error, and abend code 00E70009 is issued.

| Field | Required or optional |
|---|---|
| EXPLRC1 | Required |
| EXPLRC2 | Optional |
| XAPLONWT | Required only for UPDATE and REFERENCES table privileges |
| XAPLDIAG | Optional |

### Handling return codes

Place return codes from the exit routine in the EXPL field named EXPLRC1.

***Return codes during initialization:*** EXPLRC1 must have one of the following values during initialization:

| Value | Meaning |
|---|---|
| **0** | Initialization successful |
| **12** | Unable to service request; don’t call exit again |

See “Exception processing” for an explanation of how the EXPLRC1 value affects DB2 processing.

***Return codes during termination:*** DB2 does not check EXPLRC1 on return from the exit routine.

***Return codes during authorization check:*** Make sure that EXPLRC1 has one of the following values during the authorization check:

| Value | Meaning |
|---|---|
| **0** | Access permitted |
| **4** | Unable to determine; perform DB2 authorization checking |
| **8** | Access denied |
| **12** | Unable to service request; don’t call exit again |

See “Exception processing” for an explanation of how the EXPLRC1 value affects DB2 processing. On authorization failures, the return code is included in the IFCID 0140 trace record.

## Handling reason codes

***Reason codes during initialization:*** The reason code (EXPLRC2) that the exit routine returns after initialization determines how DB2 processes the return code (EXPLRC1) that the exit returns during initialization and authorization checking. See “Exception processing” for details.

| Value | Meaning |
|---|---|
| **–1** | Identifies the default exit routine shipped with DB2. If you replace or modify the default exit, you should not use this value. |
| **16** | Indicates to DB2 that it should terminate if the exit routine returns EXPLRC1=12, an invalid EXPLRC1 or abnormally terminates during initialization or authorization checking. ***When the exit sets the reason code to 16, DB2 does an immediate shutdown, without waiting for tasks to end.*** For long-running tasks, an immediate shutdown can mean that recovery times are long. |
| **Other** | Ignored by DB2. |

***Reason codes during authorization check:*** Field EXPLRC2 lets you put in any code that would be of use in determining why the authorization check in the exit routine failed. On authorization failures, the reason code is included in the IFCID 0140 trace record.

***Exception processing:*** During initialization or authorization checking, DB2 issues diagnostic message DSNX210I to the operator’s console, if one of the following conditions occur:

- The authorization exit returns a return code of 12 or an invalid return code.
- The authorization exit abnormally terminates.

Additional actions that DB2 performs depend on the reason code that the exit returns during initialization. Table 149 on page 921 summarizes these actions.

*Table 149. How an error condition affects DB2 actions during initialization and authorization checking*

| Exit Result | Reason code of 16 returned by exit during initialization | Reason code other than 16 or −1 returned by exit during initialization[1] |
|---|---|---|
| Return code 12 | • The task[2] abnormally terminates with reason code 00E70015<br>• DB2 terminates | • The task[2] abnormally terminates with reason code 00E70009<br>• DB2 switches to DB2 authorization checking |
| Invalid return code | • The task[2] abnormally terminates with reason code 00E70015<br>• DB2 terminates | • The task[2] abnormally terminates with reason code 00E70009<br>• DB2 switches to DB2 authorization checking |
| Abnormal termination | DB2 terminates | DB2 switches to DB2 authorization checking |

**Notes:**

1. During initialization, DB2 sets a value of −1 to identify the default exit. The user exit should not set the reason code to −1.
2. During initialization, the task is DB2 startup. During authorization checking, the task is the application.

## Debugging your exit routine

You can use IFCID 0314 to provide a trace record of the parameter list on return from the exit routine. You can activate this trace by turning on performance trace class 22.

## Determining if the exit routine is active

To determine whether the exit routine or DB2 is performing authorization checks, follow these steps:

1. Start audit trace class 1.
2. Choose a DB2 table on which to execute a SELECT statement and an authorization ID to perform the SELECT. The authorization ID must not have the DB2 SELECT privilege on the table or the external security system SELECT privilege on the table.
3. Use the authorization ID to execute a SELECT statement on the table. The SELECT statement should fail.
4. Format the trace data and examine the return code (QW0140RC) in the IFCID 0140 trace record.
   - QW0140RC = –1 indicates that DB2 performed the authorization check and denied access.
   - QW0140RC = 8 indicates that the external security system performed the authorization check and denied access.

# Edit routines

Edit routines are assigned to a table by the EDITPROC clause of CREATE TABLE. An edit routine receives the entire row of the base table in internal DB2 format; it can transform that row when it is stored by an INSERT or UPDATE SQL statement, or by the LOAD utility. It also receives the transformed row during retrieval operations and must change it back to its original form. Typical uses are to compress the storage representation of rows to save space on DASD and to encrypt the data.

You cannot use an edit routine on a table that contains a LOB or a ROWID column.

The transformation your edit routine performs on a row (possibly encryption or compression) is called *edit-encoding*. The same routine is used to undo the transformation when rows are retrieved; that operation is called *edit-decoding*.

> **Attention**
>
> The edit-decoding function must be the exact inverse of the edit-encoding function. For example, if a routine encodes 'ALABAMA' to '01', it must decode '01' to 'ALABAMA'. A violation of this rule can lead to an abend of the DB2 connecting thread, or other undesirable effects.

Your edit routine can encode the entire row of the table, including any index keys. However, index keys are extracted from the row before the encoding is done, therefore, index keys are stored in the index in *edit-decoded* form. Hence, for a table with an edit routine, index keys in the table *are* edit-coded; index keys in the index are *not* edit-coded.

The sample application contains a sample edit routine, DSN8EAE1. To print it, use ISPF facilities, IEBPTPCH, or a program of your own. Or, assemble it and use the assembly listing.

There is also a sample routine that does Huffman data compression, DSN8HUFF in library *prefix*.SDSNSAMP. That routine not only exemplifies the use of the exit parameters, it also has potentially some use for data compression. If you intend to use the routine in any production application, please pay particular attention to the warnings and restrictions given as comments in the code. You might prefer to let DB2 compress your data. For instructions, see “Compressing your data” on page 606.

## General considerations

“General considerations for writing exit routines” on page 950 applies to edit routines.

## Specifying the routine

To name an edit routine for a table, use the EDITPROC clause of the CREATE TABLE statement, followed by the name of the routine. If you plan to use an edit routine, specify it when you create the table. In operation, the routine is loaded on demand.

You cannot add an edit routine to a table that already exists: you must drop the table and re-create it. Also, you cannot alter a table with an edit routine to add a column. Again, you must drop the table and re-create it, and presumably also alter the edit routine in some way to account for the new column.

## When exits are taken

An edit routine is invoked to edit-code a row whenever DB2 inserts or updates one, including inserts made by the LOAD utility. It is invoked *after* any date routine, time routine, or field procedure. If there is also a validation routine, the edit routine is invoked *after* the validation routine. Any changes made to the row by the edit routine do not change entries made in an index.

The same edit routine is invoked to edit-decode a row whenever DB2 retrieves one. On retrieval, it is invoked *before* any date routine, time routine, or field procedure. If retrieved rows are sorted, the edit routine is invoked *before* the sort. An edit routine is not invoked for a DELETE operation without a WHERE clause that deletes an entire table in a segmented table space.

## Parameter lists on entry

At invocation, registers are set as described in “Registers at invocation” on page 951, and the edit routine uses the standard exit parameter list (EXPL) described there. Table 150 shows the exit-specific parameter list, described by macro DSNDEDIT. Figure 132 on page 924 shows how the parameter list points to other row information.

*Table 150. Parameter list for an edit routine*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| EDITCODE | 0 | Signed 4-byte integer | Edit code telling the type of function to be performed, as follows:<br>**0** Edit-encode row for insert or update<br>**4** Edit-decode row for retrieval |
| EDITROW | 4 | Address | Address of a row description. Its format is shown in Table 161 on page 954. |
|  | 8 | Signed 4-byte integer | Reserved |
| EDITILTH | C | Signed 4-byte integer | Length of the input row |
| EDITIPTR | 10 | Address | Address of the input row |
| EDITOLTH | 14 | Signed 4-byte integer | Length of output row. On entry, this is the size of the area in which to place the output row. The exit must not modify storage beyond this length. |
| EDITOPTR | 18 | Address | Address of the output row |

## Processing requirements

Your routine must be based on the DB2 data formats; see “Row formats for edit and validation routines” on page 952.

## Incomplete rows

Sometimes DB2 passes, to an edit routine, an input row that has fewer fields than there are columns in the table. In that case, the routine must stop processing the row after the last input field. Columns for which no input field is provided are always at the end of the row and are never defined as NOT NULL; either they allow nulls, they are defined as NOT NULL WITH DEFAULT, or the column is a ROWID column.

Use macro DSNDEDIT to get the starting address and row length for edit exits. Add the row length to the starting address to get the first invalid address beyond the end of the input buffer; your routine must *not* process any address as large as that.

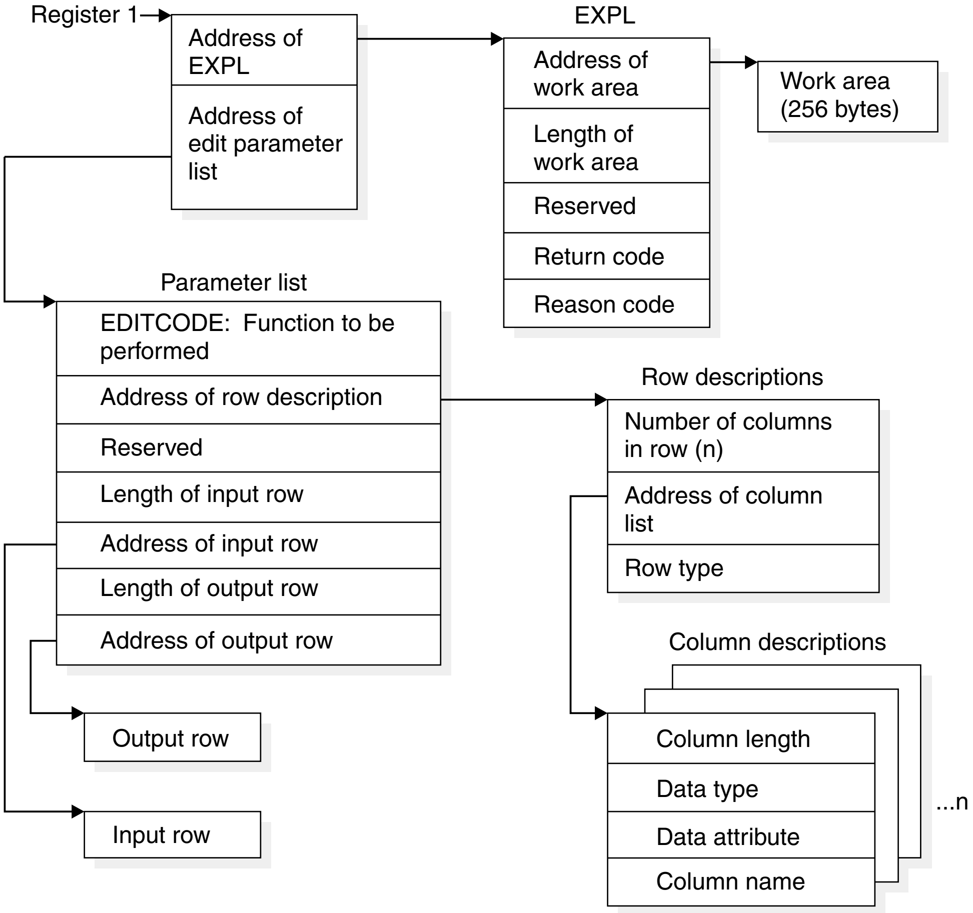


*Figure 132. How the edit exit parameter list points to row information. The address of the nth column description is given by: RFMTAFLD + (n−1)×(FFMTE−FFMT); see “Parameter list for row format descriptions” on page 954.*

## Expected output

**If EDITCODE contains 0**, the input row is in decoded form. Your routine must encode it.

In that case, the maximum length of the output area, in EDITOLTH, is 10 bytes more than the maximum length of the record. In counting the maximum length, “record” includes fields for the lengths of VARCHAR and VARGRAPHIC columns, and for null indicators, but does not include the 6-byte record header.

**If EDITCODE contains 4**, the input row is in coded form. Your routine must decode it.

In that case, EDITOLTH contains the maximum length of the record. As before, “record” includes fields for the lengths of VARCHAR and VARGRAPHIC columns, and for null indicators, but not the 6-byte record header.

**In either case**, put the result in the output area, pointed to by EDITOPTR, and put the length of your result in EDITOLTH. The length of your result must not be greater than the length of the output area, as given in EDITOLTH on invocation, and your routine must not modify storage beyond the end of the output area.

***Required return code:*** Your routine must also leave a return code in EXPLRC1, with the following meanings:

| Value | Meaning |
|---|---|
| **0** | Function performed successfully. |
| **Nonzero** | Function failed. |

If the function fails, the routine might also leave a reason code in EXPLRC2. DB2 returns SQLCODE -652 (SQLSTATE ’23506’) to the application program and puts the reason code in field SQLERRD(6) of the SQL communication area (SQLCA).

# Validation routines

Validation routines are assigned to a table by the VALIDPROC clause of CREATE TABLE and ALTER TABLE. A validation routine receives an entire row of a base table as input, and can return an indication of whether or not to allow a following INSERT, UPDATE, or DELETE operation. Typically, a validation routine is used to impose limits on the information that can be entered in a table; for example, allowable salary ranges, perhaps dependent on job category, for the employee sample table.

Although VALIDPROCs can be specified for a table that contains a LOB column, the LOB values are not passed to the validation routine. The indicator column takes the place of the LOB column.

The return code from a validation routine is checked for a 0 value before any insert, update, or delete is allowed.

## General considerations

“General considerations for writing exit routines” on page 950 applies to validation routines.

## Specifying the routine

To name a validation routine for a table, use the VALIDPROC clause of the CREATE TABLE or ALTER TABLE statement, followed by the name of the routine. In operation, the routine is loaded on demand.

You can add a validation routine to a table that is already in existence, but it is not invoked to validate data already in the table. For suggestions about existing data, see “Checking rows of a table with a new validation routine” on page 64. You can also cancel any validation routine for a table, by using VALIDPROC NULL in an ALTER TABLE statement.

## When exits are taken

A validation routine for a table is invoked when DB2 inserts or updates a row, including inserts made by the LOAD utility. The routine is invoked for most delete operations, but NOT for a mass delete of all the rows of a table made by a DELETE statement without a WHERE clause. If there are other exit routines, the validation routine is invoked *before* any edit routine, and *after* any date routine, time routine, or field procedure.

## Parameter lists on entry

At invocation, registers are set as described in “Registers at invocation” on page 951, and the validation routine uses the standard exit parameter list (EXPL) described there. Table 151 on page 926 shows the exit-specific parameter list, described by macro DSNDRVAL.

*Table 151. Parameter List for a Validation Routine*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| | 0 | Signed 4-byte integer | Reserved |
| RVALROW | 4 | Address | Address of a row description. The format of the row description is shown in Table 161 on page 954. |
| | 8 | Signed 4-byte integer | Reserved |
| RVALROWL | C | Signed 4-byte integer | Length of the input row to be validated |
| RVALROWP | 10 | Address | Address of the input row to be validated |
| | 14 | Signed 4-byte integer | Reserved |
| | 18 | Signed 4-byte integer | Reserved |
| RVALPLAN | 1C | Character, 8 bytes | Name of the plan issuing the request |
| RVALOPER | 24 | Unsigned 1-byte integer | Code identifying the operation being performed, as follows:<br>**1** Insert, update, or load<br>**2** Delete |
| RVALFL1 | 25 | Character, 1 byte | The high-order bit is on if the requester has installation SYSADM authority. The remaining 7 bits are reserved. |
| RVALCSTC | 26 | Character, 2 bytes | Connection system type code. Values are defined in macro DSNDCSTC. |

## Processing requirements

Your routine must be based on the DB2 data formats; see “Row formats for edit and validation routines” on page 952.

## Incomplete rows

Sometimes DB2 passes, to a validation routine, an input row that has fewer fields than there are columns in the table. In that case, the routine must stop processing the row after the last input field. Columns for which no input field is provided are always at the end of the row and are never defined as NOT NULL; either they allow nulls, they are defined as NOT NULL WITH DEFAULT, or the column is a ROWID column.

Use macro DSNDRVAL to get the starting address and row length for validation exits. Add the row length to the starting address to get the first invalid address beyond the end of the input buffer; your routine must *not* process any address as large as that.

## Expected output

Your routine must leave a return code in EXPLRC1, with the following meanings:

**Value** **Meaning**

| | |
|---|---|
| **0** | Allow insert, update, or delete |
| **Nonzero** | Do not allow insert, update, or delete |

If the operation is not allowed, the routine might also leave a reason code in EXPLRC2. DB2 returns SQLCODE -652 (SQLSTATE '23506') to the application program and puts the reason code in field SQLERRD(6) of the SQL communication area (SQLCA).

Figure 133 shows how the parameter list points to other information.

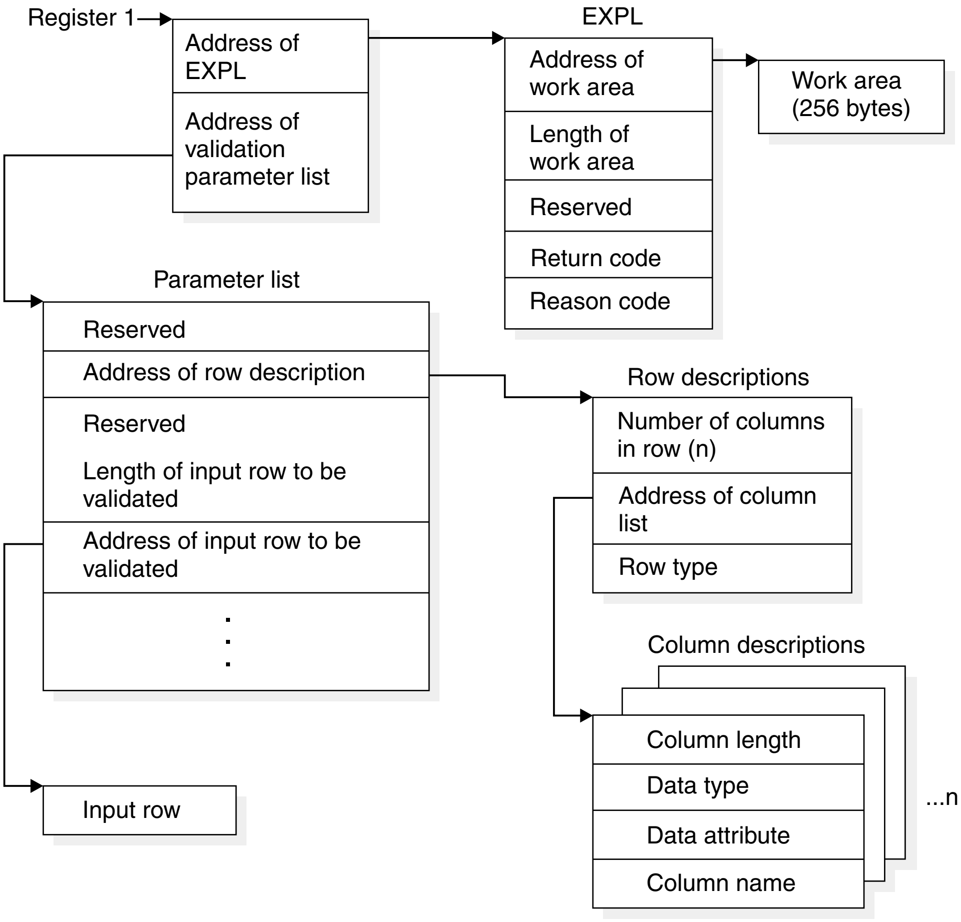


*Figure 133. How a validation parameter list points to information. The address of the nth column description is given by: RFMTAFLD + (n−1)×(FFMTE−FFMT); see "Parameter list for row format descriptions" on page 954.*

## Date and time routines

A date routine is a user-written exit routine to change date values from a locally-defined format into a format recognized by DB2, when loading or inserting them into a column with data type DATE; and from the ISO format into the locally-defined format, when retrieving the values and assigning them to a host variable. Similarly, a time routine changes time values from a locally-defined format into one recognized by DB2, and from ISO into the locally-defined format. The following table shows the formats recognized by DB2:

*Table 152. Date and Time Formats*

| Format name | Abbreviation | Typical date | Typical time |
|---|---|---|---|
| IBM European standard | EUR | 25.12.1992 | 13.30.05 |
| International Standards Organization | ISO | 1992-12-25 | 13.30.05 |
| Japanese Industrial Standard Christian Era | JIS | 1992-12-25 | 13:30:05 |
| IBM USA standard | USA | 12/25/1992 | 1:30 PM |

For an example of the use of an exit routine, suppose you want to insert and retrieve dates in a format like “September 21, 1992”. You might have a date routine that transforms that date to a format recognized by DB2—say ISO, “1992-09-21”—on insertion, and transforms “1992-09-21” to “September 21, 1992” on retrieval.

You can have either a date routine, a time routine, or both. These routines do not apply to timestamps. Both types of routine follow the rules given below. Special rules apply if you execute queries at a remote DBMS, through the distributed data facility; for that case, see Chapter 2 of *DB2 SQL Reference*.

## General considerations

“General considerations for writing exit routines” on page 950 applies to date and time routines.

## Specifying the routine

**To establish a date or time routine**, set LOCAL DATE LENGTH or LOCAL TIME LENGTH, when installing DB2, to the length of the longest field required to hold a date or time in your local format. Allowable values range from 10 to 254. For example, if you intend to insert and retrieve dates in the form “September 21, 1992”, then you need an 18-byte field. Set LOCAL DATE LENGTH to 18.

Also, replace the IBM-supplied exit routines, using CSECTs DSNXVDTX for a date routine and DSNXVTMX for a time routine. The routines are loaded when DB2 starts.

**To make the local date or time format the default for retrieval**, set DATE FORMAT or TIME FORMAT to LOCAL when installing DB2. That has the effect that DB2 *always* takes the exit routine when you retrieve from a DATE or TIME column. In our example, suppose that you want to retrieve dates in your local format only occasionally; most of the time you use the USA format. Set DATE FORMAT to USA.

The install parameters for LOCAL DATE LENGTH, LOCAL TIME LENGTH, DATE FORMAT, and TIME FORMAT can also be updated after DB2 is installed. For instructions, see Part 2 of *DB2 Installation Guide*. If you change a length parameter, you may have to rebind applications.

## When exits are taken

***On insertion:*** A date or time routine is invoked to change a value from the locally-defined format to a format recognized by DB2 in the following circumstances:

- When a date or time value is entered by an INSERT or UPDATE statement, or by the LOAD utility

- When a constant or host variable is compared to a column with a data type of DATE, TIME, or TIMESTAMP
- When the DATE or TIME scalar function is used with a string representation of a date or time in LOCAL format
- When a date or time value is supplied for a limit of a partitioned index in a CREATE INDEX statement

The exit is taken before any edit or validation routine.

- **If the default is LOCAL**, DB2 takes the exit immediately. If the exit routine does not recognize the data (EXPLRC1=8), DB2 then tries to interpret it as a date or time in one of the recognized formats (EUR, ISO JIS, or USA). DB2 rejects the data only if that interpretation also fails.
- **If the default is not LOCAL**, DB2 first tries to interpret the data as a date or time in one of the recognized formats. If that interpretation fails, DB2 then takes the exit routine, if it exists.

DB2 checks that the value supplied by the exit routine represents a valid date or time in some recognized format, and then converts it into an internal format for storage or comparison. If the value is entered into a column that is a key column in an index, the index entry is also made in the internal format.

***On retrieval:*** A date or time routine can be invoked to change a value from ISO to the locally-defined format when a date or time value is retrieved by a SELECT or FETCH statement. If LOCAL is the default, the routine is always invoked unless overridden by a precompiler option or by the CHAR function, as by specifying CHAR(HIREDATE, ISO); that specification always retrieves a date in ISO format. If LOCAL is not the default, the routine is invoked only when specifically called for by CHAR, as in CHAR(HIREDATE, LOCAL); that always retrieves a date in the format supplied by your date exit routine.

On retrieval, the exit is invoked after any edit routine or DB2 sort. A date or time routine is not invoked for a DELETE operation without a WHERE clause that deletes an entire table in a segmented table space.

## Parameter lists on entry

At invocation, registers are set as described in “Registers at invocation” on page 951, and the date or time routine uses the standard exit parameter list (EXPL) described there. Table 153 shows its exit-specific parameter list, described by macro DSNDDTXP.

*Table 153. Parameter list for a date or time routine*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| DTXPFN | 0 | Address | Address of a 2-byte integer containing a function code. The codes and their meanings are:<br>**4** Convert from local format to ISO.<br>**8** Convert from ISO to local format. |
| DTXPLN | 4 | Address | Address of a 2-byte integer containing the length in bytes of the local format. This is the length given as LOCAL DATE LENGTH or LOCAL TIME LENGTH when installing DB2. |
| DTXPLOC | 8 | Address | Address of the date or time value in local format |

*Table 153. Parameter list for a date or time routine (continued)*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| DTXPISO | C | Address | Address of the date or time value in ISO format (DTXPISO). The area pointed to is 10 bytes long for a date, 8 bytes for a time. |

## Expected output

**If the function code is 4**, the input value is in local format, in the area pointed to by DTXPLOC. Your routine must change it to ISO, and put the result in the area pointed to by DTXPISO.

**If the function code is 8**, the input value is in ISO, in the area pointed to by DTXPISO. Your routine must change it to your local format, and put the result in the area pointed to by DTXPLOC.

Your routine must also leave a return code in EXPLRC1, a 4-byte integer and the third word of the EXPL area. The return code has the following meanings:

| Value | Meaning |
|---|---|
| **0** | No errors; conversion was completed. |
| **4** | Invalid date or time value. |
| **8** | Input value not in valid format; if the function is insertion, and LOCAL is the default, DB2 next tries to interpret the data as a date or time in one of the recognized formats (EUR, ISO, JIS, or USA). |
| **12** | Error in exit routine. |

Figure 134 shows how the parameter list points to other information.

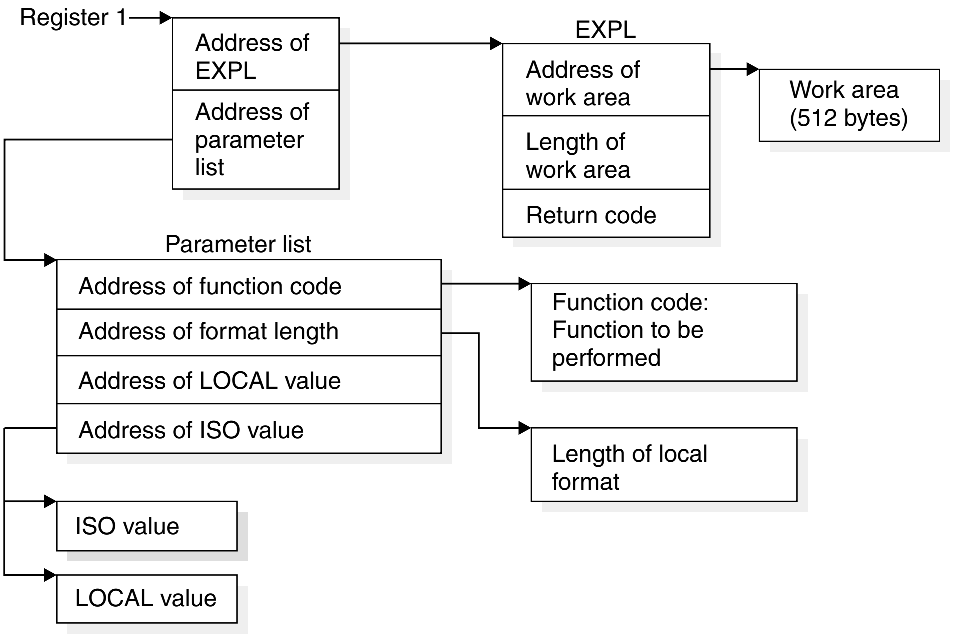


*Figure 134. How a Date or Time Parameter List Points to Other Information*

# Conversion procedures

A conversion procedure is a user-written exit routine that converts characters from one coded character set to another coded character set. (For a general discussion of character sets, and definitions of those terms, see Appendix A of *DB2 Installation Guide*.) In most cases, any conversion that is needed can be done by routines provided by IBM. The exit for a user-written routine is available to handle exceptions.

## General considerations

"General considerations for writing exit routines" on page 950 applies to conversion routines.

## Specifying the routine

**To establish a conversion procedure**, insert a row into the catalog table SYSIBM.SYSSTRINGS. The row must contain values for the following columns:

**INCCSID**
The coded character set identifier (CCSID) of the source string.

**OUTCCSID**
The CCSID of the converted string.

**TRANSTYPE**
The nature of the conversion. Values can be:

| | |
|---|---|
| **GG** | ASCII GRAPHIC to EBCDIC GRAPHIC |
| **MM** | EBCDIC MIXED to EBCDIC MIXED |
| **MP** | EBCDIC MIXED to ASCII MIXED |
| **MS** | EBCDIC MIXED to EBCDIC SBCS |
| **PM** | ASCII MIXED to EBCDIC MIXED |
| **PP** | ASCII MIXED to ASCII MIXED |
| **PS** | ASCII MIXED to EBCDIC SBCS |
| **SM** | EBCDIC SBCS to EBCDIC MIXED |
| **SP** | SBCS (ASCII or EBCDIC) to ASCII MIXED |
| **SS** | EBCDIC SBCS to EBCDIC SBCS |

**TRANSPROC**
The name of your conversion procedure.

**IBMREQD**
Must be N.

DB2 does not use the following columns, but checks them for the allowable values listed. Values you insert can be used by your routine in any way. If you insert no value in one of these columns, DB2 inserts the default value listed.

**ERRORBYTE**
Any character, or null. Default: null.

**SUBBYTE**
Any character not equal to the value of ERRORBYTE, or null. Default: null.

**TRANSTAB**
Any character string of length 256 or the empty string. Default: the empty string.

## When exits are taken

The exit is taken, and your procedure invoked, whenever a conversion is required from the coded character set identified by INCCSID to the coded character set identified by OUTCCSID.

## Parameter lists on entry

At invocation, registers are set as described in “Registers at invocation” on page 951, and the conversion procedure uses the standard exit parameter list (EXPL) described there. A conversion procedure does *not* use an exit-specific parameter list, as described in “Parameter lists” on page 951. Instead, the area pointed to by register 1 at invocation includes three words, which contain the addresses of the following items:

1. The EXPL parameter list
2. A string value descriptor, described below, that contains the character string to be converted
3. A copy of a row from SYSIBM.SYSSTRINGS, described below, that names the conversion procedure identified in TRANSPROC.

The length of the work area pointed to by the exit parameter list is generally 512 bytes. However, if the string to be converted is ASCII MIXED data (the value of TRANSTYPE in the row from SYSSTRINGS is PM or PS), then the length of the work area is 256 bytes, plus the length attribute of the string.

***The string value descriptor:*** The descriptor has the format shown in Table 154.

*Table 154. Format of string value descriptor for a conversion procedure*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| FPVDTYPE | 0 | Signed 2-byte integer | Data type of the value:<br>**Code** **Means**<br>**20** VARCHAR<br>**28** VARGRAPHIC |
| FPVDVLEN | 2 | Signed 2-byte integer | The maximum length of the string |
| FPVDVALE | 4 | None | The string. The first halfword is the string's actual length in characters. If the string is ASCII MIXED data, it is padded out to the maximum length by undefined bytes. |

***The row from SYSSTRINGS:*** The row copied from the catalog table SYSIBM.SYSSTRINGS is in the standard DB2 row format described in “Row formats for edit and validation routines” on page 952. The fields ERRORBYTE and SUBBYTE each include a null indicator. The field TRANSTAB is of varying length and begins with a 2-byte length field.

## Expected output

Except in the case of certain errors, described below, your conversion procedure should replace the string in FPVDVALE with the converted string. When converting MIXED data, your procedure must ensure that the result is well-formed. In any conversion, if you change the length of the string, you must set the length control field in FPVDVALE to the proper value. Over-writing storage beyond the maximum length of the FPVDVALE causes an abend.

Your procedure must also set a return code in field EXPLRC1 of the exit parameter list, as shown below.

With these two codes, provide the converted string in FPVDVALE:

| **Code** | **Meaning** |
|---|---|
| **0** | Successful conversion |
| **4** | Conversion with substitution |

For the remaining codes, DB2 does not use the converted string:

| **Code** | **Meaning** |
|---|---|
| **8** | Length exception |
| **12** | Invalid code point |
| **16** | Form exception |
| **20** | Any other error |
| **24** | Invalid CCSID |

***Exception conditions:*** Return a length exception (code 8) when the converted string is longer than the maximum length allowed.

For an invalid code point (code 12), place the 1- or 2-byte code point in field EXPLRC2 of the exit parameter list.

Return a form exception (code 16) for EBCDIC MIXED data when the source string does not conform to the rules for MIXED data.

Any other uses of codes 8 and 16, or of EXPLRC2, are optional.

***Error conditions:*** On return, DB2 considers any of the following conditions as a “conversion error”:

- EXPLRC1 is greater than 16.
- EXPLRC1 is 8, 12, or 16 and the operation that required the conversion is *not* an assignment of a value to a host variable with an indicator variable.
- FPVDTYPE or FPVDVLEN has been changed.
- The length control field of FPVDVALE is greater than the original value of FPVDVLEN or is negative.

In the case of a conversion error, DB2 sets the SQLERRMC field of the SQLCA to HEX(EXPLRC1) CONCAT X'FF' CONCAT HEX(EXPLRC2).

Figure 135 shows how the parameter list points to other information.

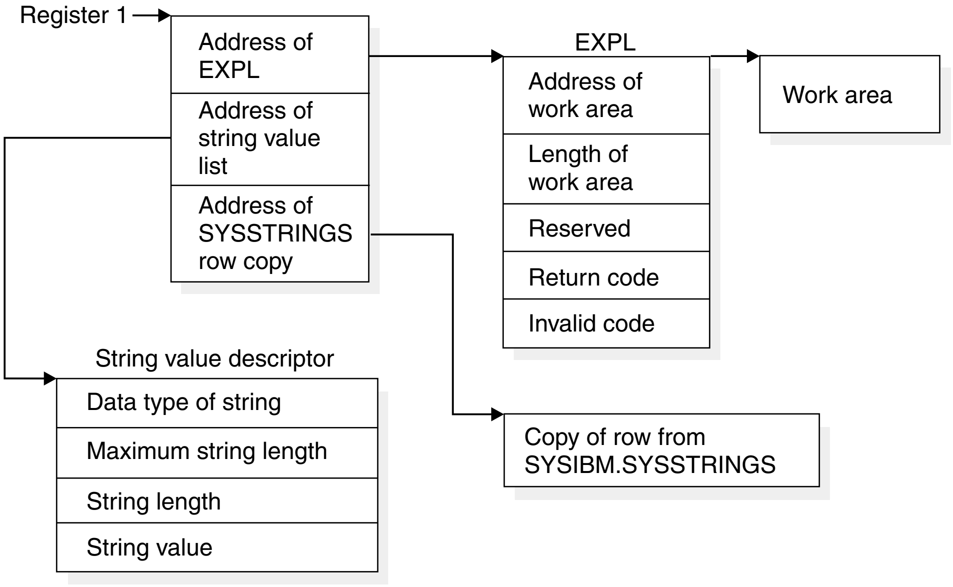


*Figure 135. Pointers at entry to a conversion procedure*

## Field procedures

Field procedures are assigned to a table by the FIELDPROC clause of CREATE TABLE and ALTER TABLE. A field procedure is a user-written exit routine to transform values in a single short-string column. When values in the column are changed, or new values inserted, the field procedure is invoked for each value, and can transform that value (encode it) in any way. The encoded value is then stored. When values are retrieved from the column, the field procedure is invoked for each value, which is encoded, and must decode it back to the original string value.

Any indexes, including partitioned indexes, defined on a column that uses a field procedure are built with encoded values. For a partitioned index, the encoded value of the limit key is put into the LIMITKEY column of the SYSINDEXPART table. Hence, a field procedure might be used to alter the sorting sequence of values entered in a column. For example, telephone directories sometimes require that names like “McCabe” and “MacCabe” appear next to each other, an effect that the standard EBCDIC sorting sequence does not provide. And languages that do not use the Roman alphabet have similar requirements. However, if a column is provided with a suitable field procedure, it can be correctly ordered by ORDER BY.

The transformation your field procedure performs on a value is called *field-encoding*. The same routine is used to undo the transformation when values are retrieved; that operation is called *field-decoding*. Values in columns with a field procedure are described to DB2 in two ways:

1. The description of the column as defined in CREATE TABLE or ALTER TABLE appears in the catalog table SYSIBM.SYSCOLUMNS. That is the description of the field-decoded value, and is called the *column description*.
2. The description of the encoded value, as it is stored in the data base, appears in the catalog table SYSIBM.SYSFIELDS. That is the description of the field-encoded value, and is called the *field description*.

**Attention:** The field-decoding function must be the exact inverse of the field-encoding function. For example, if a routine encodes 'ALABAMA' to '01', it must decode '01' to 'ALABAMA'. A violation of this rule can lead to an abend of the DB2 connecting thread, or other undesirable effects.

## Field definition

The field procedure is also invoked when the table is created or altered, to define the data type and attributes of an encoded value to DB2; that operation is called *field-definition*. The data type of the encoded value can be any valid SQL data type except DATE, TIME, TIMESTAMP, LONG VARCHAR, or LONG VARGRAPHIC; the allowable types are listed in the description of field FPVDTYPE in Table 157 on page 939. The length, precision, or scale of the encoded value must be compatible with its data type.

A user-defined data type can be a valid field if the source type of the data type is a short string column that has a null default value. DB2 casts the value of the column to the source type before it passes it to the field procedure.

## General considerations

"General considerations for writing exit routines" on page 950 applies to field procedures.

## Specifying the procedure

To name a field procedure for a column, use the FIELDPROC clause of the CREATE TABLE or ALTER TABLE statement, followed by the name of the procedure and, optionally, a list of parameters. You can use a field procedure only with a short string column. You cannot use a field procedure on a column defined using NOT NULL WITH DEFAULT.

If you plan to use a field procedure, specify it when you create the table. In operation, the procedure is loaded on demand. You cannot add a field procedure to an existing column of a table; you can, however, use ALTER TABLE to add to an existing table a new column that uses a field procedure.

You cannot use a field procedure on a LOB or a ROWID column. Field procedures can be specified for other columns of a table that contains a LOB or ROWID column.

The optional parameter list that follows the procedure name is a list of constants, enclosed in parentheses, called the *literal list*. The literal list is converted by DB2 into a data structure called the *field procedure parameter value list* (FPPVL). That structure is passed to the field procedure during the field-definition operation. At that time, the procedure can modify it or return it unchanged. The output form of the FPPVL we call the *modified FPPVL*; it is stored in the DB2 catalog as part of the field description. The modified FPPVL is passed again to the field procedure whenever that procedure is invoked for field-encoding or field-decoding.

## When exits are taken

A field procedure specified for a column is invoked in three general situations:

1. **For field-definition**, when the CREATE TABLE or ALTER TABLE statement that names the procedure is executed. During this invocation, the procedure is expected to:
   - Determine whether the data type and attributes of the column are valid.
   - Verify the literal list, and change it if wanted.
   - Provide the field description of the column.

- Define the amount of working storage needed by the field-encoding and field-decoding processes.

2. **For field-encoding**, when a column value is to be field-encoded. That occurs for any value that:
   - Is inserted in the column by an SQL INSERT statement, or loaded by the DB2 LOAD utility.
   - Is changed by an SQL UPDATE statement.
   - Is compared to a column with a field procedure, unless the comparison operator is LIKE. The value being encoded is a host variable or constant. (When the comparison operator is LIKE, the column value is decoded.)
   - Defines the limit of a partition of an index. The value being encoded follows VALUES in the PART clause of CREATE INDEX.

   If there are any other exit routines, the field procedure is invoked *before* any of them.
3. **For field-decoding**, when a stored value is to be field-decoded back into its original string value. This occurs for any value that is:
   - Retrieved by an SQL SELECT or FETCH statement, or by the unload phase of the REORG utility.
   - Compared to another value with the LIKE comparison operator. The value being decoded is from the column that uses the field procedure.

   In this case, the field procedure is invoked *after* any edit routine or DB2 sort.

A field procedure is never invoked to process a null value, nor for a DELETE operation without a WHERE clause on a table in a segmented table space.

**A warning about blanks:** When DB2 compares the values of two strings with different lengths, it temporarily pads the shorter string with blanks (in EBCDIC or double-byte characters, as needed) up to the length of the longer string. If the shorter string is the value of a column with a field procedure, the padding is done to the encoded value, but the pad character is *not* encoded. Therefore, if the procedure changes blanks to some other character, encoded blanks at the end of the longer string are not equal to padded blanks at the end of the shorter string. That situation can lead to errors; for example, some strings that ought to be equal might not be recognized as such. Therefore, we recommend *not* encoding blanks by a field procedure.

## Control blocks for execution

This section describes certain control blocks that are used to communicate to a field procedure, under the following headings:
“The field procedure parameter list (FPPL)” on page 937
“The work area” on page 937
“The field procedure information block (FPIB)” on page 937
“The field procedure parameter value list (FPPVL)” on page 938
“Value descriptors” on page 938.

Following that are the specific requirements for the three operations of field-definition:
“Field-definition (function code 8)” on page 939
“Field-encoding (function code 0)” on page 941
“Field-decoding (function code 4)” on page 943.

The contents of registers at invocation and at exit are different for each of those operations, and are described with the requirements for the operations.

## The field procedure parameter list (FPPL)

The field procedure parameter list is pointed to by register 1 on entry to a field procedure. It, in turn, contains the addresses of five other areas, as shown in Figure 136. Those areas are described in the following pages. The FPPL and the areas it points to are all described by the mapping macro DSNDFPPB.

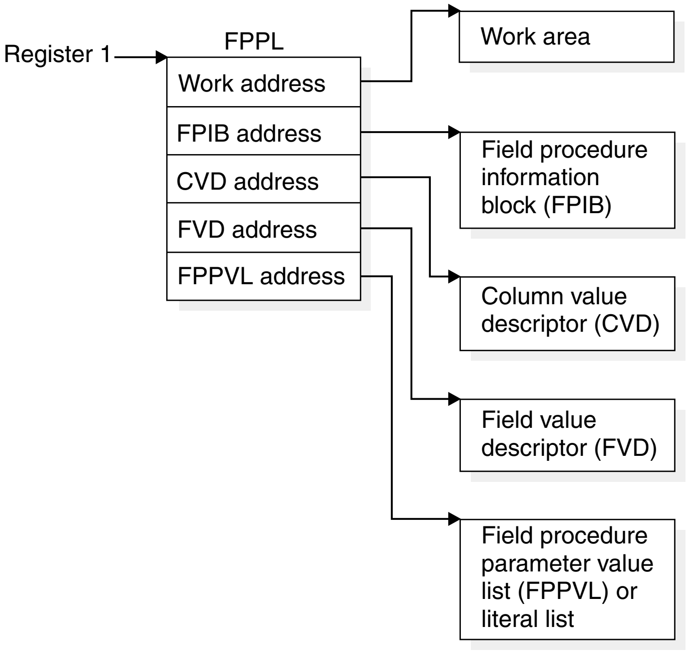

Register 1
FPPL
Work address
FPIB address
CVD address
FVD address
FPPVL address
Work area
Field procedure information block (FPIB)
Column value descriptor (CVD)
Field value descriptor (FVD)
Field procedure parameter value list (FPPVL) or literal list

*Figure 136. Field procedure parameter list*

## The work area

The work area is a contiguous, uninitialized area of locally-addressable, pageable, swappable, fetch-protected storage, obtained in storage key 7 and subpool 229. The area can be used by a field procedure as working storage. A new area is provided each time the procedure is invoked.

The size of the area you need depends on the way you have programmed your field-encoding and field-decoding operations. Suppose, for example, that the longest work area you need for either of those operations is 1024 bytes. DB2 passes to your routine, for the field-definition operation, a value of 512 bytes for the length of the work area; your field-definition operation must change that to 1024. Thereafter, whenever your field procedure is invoked for encoding or decoding, DB2 makes available to it an area of 1024 bytes.

If 512 bytes is sufficient for your operations, your field-definition operation need not change the value supplied by DB2. If you need less than 512 bytes, your field-definition can return a smaller value.

## The field procedure information block (FPIB)

The field procedure information block communicates general information to a field procedure. For example, it tells what operation is to be done, allows the field procedure to signal errors, and gives the size of the work area.

It has the format shown in Table 155.

*Table 155. Format of FPIB, defined in copy macro DSNDFPPB*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| FPBFCODE | 0 | Signed 2-byte integer | Function code<br>**Code** **Means**<br>**0** Field-encoding<br>**4** Field-decoding<br>**8** Field-definition |
| FPBWKLN | 2 | Signed 2-byte integer | Length of work area; the maximum is 32767 bytes. |
| FPBSORC | 4 | Signed 2-byte integer | Reserved |
| FPBRTNC | 6 | Character, 2 bytes | Return code set by field procedure |
| FPBRSNCD | 8 | Character, 4 bytes | Reason code set by field procedure |
| FPBTOKPT | C | Address | Address of a 40-byte area, within the work area or within the field procedure's static area, containing an error message |

## The field procedure parameter value list (FPPVL)

The field procedure parameter value list communicates the literal list, supplied in the CREATE TABLE or ALTER TABLE statement, to the field procedure during field-definition. At that time the field procedure can reformat the FPPVL; it is the reformatted FPPVL that is stored in SYSIBM.SYSFIELDS and communicated to the field procedure during field-encoding and field-decoding as the *modified FPPVL*.

The FPPVL has the format shown in Table 156.

*Table 156. Format of FPPVL, defined in copy macro DSNDFPPB*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| FPPVLEN | 0 | Signed 2-byte integer | Length in bytes of the area containing FPPVCNT and FPPVVDS. At least 254 for field-definition. |
| FPPVCNT | 2 | Signed 2-byte integer | Number of value descriptors that follow, equal to the number of parameters in the FIELDPROC clause. Zero if no parameters were listed. |
| FPPVVDS | 4 | Structure | For each parameter in the FIELDPROC clause, there is:<br>1. A signed 4-byte integer giving the length of the following value descriptor, which includes the lengths of FPVDTYPE, FPVDLEN, and FPVDVALE.<br>2. A value descriptor |

## Value descriptors

A value descriptor describes the data type and other attributes of a value. Value descriptors are used with field procedures in these ways:

- During field-definition, they describe each constant in the field procedure parameter value list (FPPVL). The set of these value descriptors is part of the FPPVL control block.

- During field-encoding and field-decoding, the decoded (column) value and the encoded (field) value are described by the column value descriptor (CVD) and the field value descriptor (FVD).

The *column value descriptor (CVD)* contains a description of a column value and, if appropriate, the value itself. During field-encoding, the CVD describes the value to be encoded. During field-decoding, it describes the decoded value to be supplied by the field procedure. During field-definition, it describes the column as defined in the CREATE TABLE or ALTER TABLE statement.

The *field value descriptor (FVD)* contains a description of a field value and, if appropriate, the value itself. During field-encoding, the FVD describes the encoded value to be supplied by the field procedure. During field-decoding, it describes the value to be decoded. Field-definition must put into the FVD a description of the encoded value.

Value descriptors have the format shown in Table 157.

*Table 157. Format of value descriptors*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| FPVDTYPE | 0 | Signed 2-byte integer | Data type of the value:<br>**Code** **Means**<br>**0** INTEGER<br>**4** SMALLINT<br>**8** FLOAT<br>**12** DECIMAL<br>**16** CHAR<br>**20** VARCHAR<br>**24** GRAPHIC<br>**28** VARGRAPHIC |
| FPVDVLEN | 2 | Signed 2-byte integer | • For a varying-length string value, its maximum length<br>• For a decimal number value, its precision (byte 1) and scale (byte 2)<br>• For any other value, its length |
| FPVDVALE | 4 | None | The value. The value is in external format, not DB2 internal format. If the value is a varying-length string, the first halfword is the value's actual length in bytes. This field is not present in a CVD, or in an FVD used as input to the field-definition operation. An empty varying-length string has a length of zero with no data following. |

## Field-definition (function code 8)

The input provided to the field-definition operation, and the output required, are as follows:

### On ENTRY

The **registers** have the following information:

| Register | Contains |
|---|---|
| **1** | Address of the field procedure parameter list (FPPL); see Figure 136 on page 937 for a schematic diagram. |

| | |
|---|---|
| **2 through 12** | Unknown values that must be restored on exit. |
| **13** | Address of the register save area. |
| **14** | Return address. |
| **15** | Address of entry point of exit routine. |

The contents of all other registers, and of fields not listed below, are unpredictable.

The **work area** consists of 512 contiguous uninitialized bytes.

The **FPIB** has the following information:

| Field | Contains |
|---|---|
| **FPBFCODE** | 8, the function code |
| **FPBWKLN** | 512, the length of the work area |

The **CVD** has the following information:

**Field** **Contains**

**FPVDTYPE** One of these codes for the data type of the column value:

| Code | Means |
|---|---|
| **16** | CHAR |
| **20** | VARCHAR |
| **24** | GRAPHIC |
| **28** | VARGRAPHIC |

**FPVDVLEN** The length attribute of the column

The FPVDVALE field is omitted.

The **FVD** provided is 4 bytes long.

The **FPPVL** has the following information:

| Field | Contains |
|---|---|
| **FPPVLEN** | The length, in bytes, of the area containing the parameter value list. The minimum value is 254, even if there are no parameters. |
| **FPPVCNT** | The number of value descriptors that follow; zero if there are no parameters. |
| **FPPVVDS** | A contiguous set of value descriptors, one for each parameter in the parameter value list, each preceded by a 4-byte length field. |

## On EXIT

The **registers** must have the following information:

| Register | Contains |
|---|---|
| **2 through 12** | The values that they contained on entry. |
| **15** | The integer zero if the column described in the CVD is valid for the field procedure; otherwise the value must *not* be zero. |

Fields listed below must be set as shown; all other fields must remain as on entry.

The **FPIB** must have the following information:

**Field** **Contains**

| | |
|---|---|
| **FPBWKLN** | The length, in bytes, of the work area to be provided to the field-encoding and field-decoding operations; 0 if no work area is required. |
| **FPBRTNC** | An optional 2-byte character return code, defined by the field procedure; blanks if no return code is given. |
| **FPBRSNC** | An optional 4-byte character reason code, defined by the field procedure; blanks if no reason code is given. |
| **FPBTOKP** | Optionally, the address of a 40-byte error message residing in the work area or in the field procedure's static area; zeros if no message is given. |

Errors signalled by a field procedure result in SQLCODE -681 (SQLSTATE ’23507’), which is set in the SQL communication area (SQLCA). The contents of FPBRTNC and FPBRSNC, and the error message pointed to by FPBTOKP, are also placed into the tokens, in SQLCA, as field SQLERRMT. The meaning of the error message is determined by the field procedure.

The **FVD** must have the following information:

| **Field** | **Contains** |
|---|---|
| **FPVDTYPE** | The numeric code for the data type of the field value. Any of the data types listed in Table 157 on page 939 is valid. |
| **FPVDVLEN** | The length of the field value. |

Field FPVDVALE must not be set; the length of the FVD is 4 bytes only.

The **FPPVL** can be redefined to suit the field procedure, and returned as the *modified FPPVL*, subject to the following restrictions:

- The field procedure must not increase the length of the FPPVL.
- FPPVLEN must contain the actual length of the modified FPPVL, or 0 if no parameter list is returned.

The modified FPPVL is recorded in the catalog table SYSIBM.SYSFIELDS, and is passed again to the field procedure during field-encoding and field-decoding. The modified FPPVL need not have the format of a field procedure parameter list, and it need not describe constants by value descriptors.

## Field-encoding (function code 0)

The input provided to the field-encoding operation, and the output required, are as follows:

### On ENTRY

The **registers** have the following information:

| **Register** | **Contains** |
|---|---|
| **1** | Address of the field procedure parameter list (FPPL); see Figure 136 on page 937 for a schematic diagram. |
| **2 through 12** | Unknown values that must be restored on exit. |
| **13** | Address of the register save area. |
| **14** | Return address. |
| **15** | Address of entry point of exit routine. |

The contents of all other registers, and of fields not listed below, are unpredictable.

The **work area** is contiguous, uninitialized, and of the length specified by the field procedure during field-definition.

The **FPIB** has the following information:

| Field | Contains |
|---|---|
| **FPBFCODE** | 0, the function code |
| **FPBWKLN** | The length of the work area |

The **CVD** has the following information:

| Field | Contains |
|---|---|
| **FPVDTYPE** | The numeric code for the data type of the column value, as shown in Table 157 on page 939. |
| **FPVDVLEN** | The length of the column value. |
| **FPVDVALE** | The column value; if the value is a varying-length string, the first halfword contains its length. |

The **FVD** has the following information:

| Field | Contains |
|---|---|
| **FPVDTYPE** | The numeric code for the data type of the field value. |
| **FPVDVLEN** | The length of the field value. |
| **FPVDVALE** | An area of unpredictable content that is as long as the field value. |

The *modified* FPPVL, produced by the field procedure during field-definition, is provided.

## On EXIT

The **registers** have the following information:

| Register | Contains |
|---|---|
| **2 through 12** | The values that they contained on entry. |
| **15** | The integer zero if the column described in the CVD is valid for the field procedure; otherwise the value must not be zero. |

The **FVD** must contain the encoded (field) value in field FPVDVALE. If the value is a varying-length string, the first halfword must contain its length.

The **FPIB** can have the following information:

| Field | Contains |
|---|---|
| **FPBRTNC** | An optional 2-byte character return code, defined by the field procedure; blanks if no return code is given. |
| **FPBRSNC** | An optional 4-byte character reason code, defined by the field procedure; blanks if no reason code is given. |
| **FPBTOKP** | Optionally, the address of a 40-byte error message residing in the work area or in the field procedure's static area; zeros if no message is given. |

Errors signalled by a field procedure result in SQLCODE -681 (SQLSTATE ’23507’), which is set in the SQL communication area (SQLCA). The contents of FPBRTNC and FPBRSNC, and the error message pointed to by FPBTOKP, are also placed into the tokens, in SQLCA, as field SQLERRMT. The meaning of the error message is determined by the field procedure.

All other fields must remain as on entry.

## Field-decoding (function code 4)

The input provided to the field-decoding operation, and the output required, are as follows:

### On ENTRY

The **registers** have the following information:

| Register | Contains |
|---|---|
| **1** | Address of the field procedure parameter list (FPPL); see Figure 136 on page 937 for a schematic diagram. |
| **2 through 12** | Unknown values that must be restored on exit. |
| **13** | Address of the register save area. |
| **14** | Return address. |
| **15** | Address of entry point of exit routine. |

The contents of all other registers, and of fields not listed below, are unpredictable.

The **work area** is contiguous, uninitialized, and of the length specified by the field procedure during field-definition.

The **FPIB** has the following information:

| Field | Contains |
|---|---|
| **FPBFCODE** | 4, the function code |
| **FPBWKLN** | The length of the work area |

The **CVD** has the following information:

| Field | Contains |
|---|---|
| **FPVDTYPE** | The numeric code for the data type of the column value, as shown in Table 157 on page 939. |
| **FPVDVLEN** | The length of the column value. |
| **FPVDVALE** | The column value. If the value is a varying-length string, the first halfword contains its length. |

The **FVD** has the following information:

| Field | Contains |
|---|---|
| **FPVDTYPE** | The numeric code for the data type of the field value. |
| **FPVDVLEN** | The length of the field value. |
| **FPVDVALE** | The field value. If the value is a varying-length string, the first halfword contains its length. |

The *modified* FPPVL, produced by the field procedure during field-definition, is provided.

### On EXIT

The **registers** have the following information:

| Register | Contains |
|---|---|
| **2 through 12** | The values they contained on entry. |
| **15** | The integer zero if the column described in the FVD is valid for the field procedure; otherwise the value must not be zero. |

The **CVD** must contain the decoded (column) value in field FPVDVALE. If the value is a varying-length string, the first halfword must contain its length.

The **FPIB** can have the following information:

| Field | Contains |
|---|---|
| **FPBRTNC** | An optional 2-byte character return code, defined by the field procedure; blanks if no return code is given. |
| **FPBRSNC** | An optional 4-byte character reason code, defined by the field procedure; blanks if no reason code is given. |
| **FPBTOKP** | Optionally, the address of a 40-byte error message residing in the work area or in the field procedure's static area; zeros if no message is given. |

Errors signalled by a field procedure result in SQLCODE -681 (SQLSTATE ’23507’), which is set in the SQL communication area (SQLCA). The contents of FPBRTNC and FPBRSNC, and the error message pointed to by FPBTOKP, are also placed into the tokens, in SQLCA, as field SQLERRMT. The meaning of the error message is determined by the field procedure.

All other fields must remain as on entry.

# Log capture routines

A log capture exit routine makes DB2 log data available for recovery purposes in real time. The routine receives data when DB2 writes data to the active log. Your local specifications determine what the routine does with that data. The routine does not enter or return data to DB2.

**Performance warning:** Your log capture routine receives control often. Design it with care: a poorly designed routine can seriously degrade system performance. Whenever possible, use the instrumentation facility interface (IFI), rather than a log capture exit routine, to read data from the log. For instructions, see “Reading log records with IFI” on page 968.

## General considerations

“General considerations for writing exit routines” on page 950 applies, but with the following exceptions to the description of execution environments:

> A log capture routine can execute in either TCB mode or SRB mode, depending on the function it is performing. When in SRB mode, it must not perform any I/O operations nor invoke any SVC services or ESTAE routines.

## Specifying the routine

The module name for the routine is DSNJL004. Its entry point is DSNJW117.

The module is loaded during DB2 initialization and deleted during DB2 termination. You must link the module into either the *prefix*.SDSNEXIT or the DB2 *prefix*.SDSNLOAD library. Specify the REPLACE parameter of the link-edit job to replace a module that is part of the standard DB2 library for this release. The module should have attributes AMODE(31) and RMODE(ANY).

## When exits are taken

The log capture exit is taken in three possible situations, identified by a character in the exit parameter list. In two of those situations, processing operates in TCB mode;

in one situation, processing operates in SRB mode. The two modes have different processing capabilities, which your routine must be aware of. The character identifications, situations, and modes are:

- I=Initialization, Mode=TCB

  The TCB mode allows all MVS/DFP™ functions to be utilized, including ENQ, ALLOCATION, and OPEN. No buffer addresses are passed in this situation. The routine runs in supervisor state, key 7, and enabled.

  This is the *only* situation in which DB2 checks a return code from the user's log capture exit routine. The DB2 subsystem is sensitive to a return code of X'20' here. **Never return X'20'** in register 15 in this situation.

- W=Write, Mode=SRB (service request block)

  The SRB mode restricts the exit routine's processing capabilities. No supervisor call (SVC) instructions can be used, including ALLOCATION, OPEN, WTO, any I/O instruction, and so on. At the exit point, DB2 is running in supervisor state, key 7, and is enabled.

  Upon entry, the exit routine has access to buffers that have log control intervals with “blocked log records”. The first and last buffer address and control interval size fields can be used to determine how many buffers are being passed.

  See *OS/390 MVS Programming: Authorized Assembler Services Guide* for additional material on SRB-mode processing.

  **Performance warning**: All processing time required by the exit routine lengthens the time required to write the DB2 log. The DB2 address space usually has a high priority, and all work done in it in SRB mode precedes all TCB access, so any errors or long processing times can impact all DB2 processing and cause system-wide performance problems. The performance of your routine is *extremely* critical in this phase.

- T=Termination, Mode=TCB

  Processing capabilities are the same as for initialization.

A log control interval can be passed more than once. Use the time stamp to determine the last occurrence of the control interval. This last occurrence should replace all others. The time stamp is found in the control interval.

## Parameter lists on entry

At invocation, registers are set as described in “Registers at invocation” on page 951, and the log capture routine uses the standard exit parameter list (EXPL) described there. (The reason and return codes in that list can be ignored.) Table 158 shows the exit-specific parameter list; it is mapped by macro DSNDLOGX.

*Table 158. Log capture routine specific parameter list*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| LOGXEYE | 00 | Character, 4 bytes | Eye catcher: LOGX |
| LOGXLNG | 04 | Signed 2-byte integer | Length of parameter list |
| | 06 | | Reserved |
| | 08 | | Reserved |

*Table 158. Log capture routine specific parameter list (continued)*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| LOGXTYPE | 10 | Character, 1 byte | Situation identifier:<br>**I** Initialization<br>**W** Write<br>**T** Termination<br>**P** Partial control interval (CI) call |
| LOGXFLAG | 11 | Hex | Mode identifier.<br>**X'00'** SRB mode<br>**X'01'** TCB mode |
| LOGXSRBA | 12 | Character, 6 bytes | First log RBA, set when DB2 is started. The value remains constant while DB2 is active. |
| LOGXARBA | 18 | Character, 6 bytes | Highest log archive RBA used. The value is updated after completion of each log archive operation. |
|  | 1E |  | Reserved |
| LOGXRBUF | 20 | Character, 8 bytes | Range of consecutive log buffers:<br>Address of first log buffer<br>Address of last log buffer |
| LOGXBUFL | 28 | Signed 4-byte integer | Length of single log buffer (constant 4096) |
| LOGXSSID | 2C | Character, 4 bytes | DB2 subsystem id, 4 characters left justified |
| LOGXSTIM | 30 | Character, 8 bytes | DB2 subsystem startup time (TIME format with DEC option: 0CYYDDDFHHMMSSTH) |
| LOGXREL | 38 | Character, 3 bytes | DB2 subsystem release level |
| LOGXMAXB | 3B | Character, 1 byte | Maximum number of buffers that can be passed on one call. The value remains constant while DB2 is active. |
|  | 3C | 8 bytes | Reserved |
| LOGXUSR1 | 44 | Character, 4 bytes | First word of a doubleword work area for the user routine. (The content is not changed by DB2.) |
| LOGXUSR2 | 48 | Character, 4 bytes | Second word of user work area. |

## Routines for dynamic plan selection in CICS

CICS transactions can select plans dynamically by an exit routine.

***First, reconsider:*** The function was originally intended to ease two problems that can occur, for a program running under a CICS transaction, when all SQL calls are bound into a single large plan. First, changing one DBRM requires all of them to be bound again. Second, binding a large plan can be very slow, and the entire transaction is unavailable for processing during the operation. An application that is designed around small packages avoids both those problems. For guidance on using packages, see Part 5 of *DB2 Application Programming and SQL Guide*.

## What the exit routine does

Normally, the parameter PLAN=*planname* in the RCT names the plan associated with the thread for a transaction. However, if the RCT has PLNEXIT=YES, the specified exit routine names the plan dynamically.

The exit routine can name the plan during execution of the transaction at one of two times:

- When the first SQL statement in the transaction is about to be executed. That action is called *dynamic plan selection*.
- When the first SQL statement following a sync point is about to be executed, if the sync point releases a thread for reuse and if several other conditions are satisfied. That action is called *dynamic plan switching*. If you think you need that function, see particularly “Dynamic plan switching” on page 948 and then consider packages again.

## General considerations

You can specify the same exit routine for all entries in the resource control table (RCT), or different routines for different entries. You can select plans dynamically for RCT entries of both TYPE=ENTRY and TYPE=POOL.

## Execution environment

The execution environment is:

- Problem program state
- Enabled for interrupts
- PSW Key: the CICS main key for CICS 3.2 and earlier releases, or the key as specified in the CICS RDO definition ″DEFINE PROGRAM EXECKEY(USER|CICS)″.
- Non-cross-memory mode
- No MVS locks held
- Under the main TCB in the CICS address space
- 24-bit addressing mode, for any release of CICS earlier than CICS Version 4

## Specifying the routine

To specify an exit routine for dynamic plan selection, take these steps:

1. Code the routine (or use the IBM-provided sample exit routine).
2. Link-edit the routine into a load library. Concatenate that library in the DD statement DFHRPL of the JCL that initializes CICS.
3. Define the routine to CICS with resource definition on line (RDO) or by updating and re-assembling the processing program table (PPT).
4. Update the RCT with these parameters for DSNCRCT TYPE=ENTRY or TYPE=POOL:

   **PLNEXIT=**
   YES

   **PLNPGME=**
   Name of the exit routine

   Consider these parameters also for DSNCRCT TYPE=INIT:

   **PLNPGMI=**
   Name of the default exit routine for dynamic plan selection

   **PLNXTR1=**
   Integer ID for the CICS trace of entry points for plan selection

**PLNXTR2=**
Integer ID for the CICS trace of exit points for plans

For detailed information on coding those parameters, see Part 2 of *DB2 Installation Guide*.

5. Reassemble the RCT.

The exit routine can change the plan that is allocated by changing the contents of field CPRMPLAN in its parameter list. If the routine does not change the value of CPRMPLAN, the plan that is allocated has the DBRM name of the first SQL statement executed.

## Sample exit routine

A sample exit routine is available in two versions:

| **Version** | **Member and library** |
|---|---|
| **Source code** | DSNC@EXT in the CICS SDFHSAMP library |
| **Executable** | DSNCUEXT in the CICS SDFHLOAD library |

The sample routine does not change the parameter list. As a result, the name of the plan selected is, by default, the DBRM of the first SQL statement. The sample establishes addressability to the parameter list and then issues EXEC CICS RETURN.

## When exits are taken

The first SQL statement executed in a CICS transaction creates (or reuses) a thread to DB2. The dynamic plan exit is always taken at the first SQL statement in a transaction, for dynamic plan selection.

The exit can also be taken at the first SQL statement following a sync point, for dynamic plan switching. Whether the exit is taken at that time is determined by the rules described below.

## Dynamic plan switching

For you to use dynamic plan switching:

- The sync point must release the thread for reuse.
- The pool thread definition must specify PLNEXIT=YES.
- The transaction must be terminal driven.
- The transaction must use a pool thread or an unprotected entry thread that has been diverted to the pool. To use a pool thread, do not use an RCT entry, thus using TYPE=POOL as the default, or code the RCT entry in either of these ways:
  - TYPE=POOL
  - TYPE=ENTRY, THRDM=0, TWAIT=POOL
- You **must not code** either of these combinations in your RCT:
  - THRDS>0, TWAIT=POOL, and PLNEXIT=YES
  - THRDA>THRDS and PLNEXIT=YES (where THRDA and THRDS are both greater than 0)

## Coding the exit routine

An exit routine for dynamic plan selection is a user-written CICS command-level program. To use different exit routines for different RCT entries, define each routine in the RCT.

You can use the sample program, DSNC@EXT, as an example for coding your own exit routine. Your routine:

- Must adhere to normal CICS conventions for command-level programs
- Can be written in any language supported by CICS, such as assembler, COBOL, or PL/I
- Must establish addressability to the parameter list DFHCOMMAREA, using standard CICS command-level conventions
- Can update the parameter list if necessary
- Can change the plan that is allocated by changing the contents of field CPRMPLAN in the parameter list
- Must not contain SQL statements
- Must not issue the command EXEC CICS SYNCPOINT
- Must terminate by using the command EXEC CICS RETURN

## Parameter list on entry

When linking, CICS passes a parameter list to the exit routine in the CICS control block DFHCOMMAREA. Table 159 shows the contents of the list.

*Table 159. Parameter list for an exit routine for dynamic plan selection*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| CRPMPLAN | 0 | Character, 8 bytes | The DBRM or plan name for the first SQL statement to be executed after the exit routine. The routine can change this field to establish a new plan. |
| CPRMAUTH | 8 | Character, 8 bytes | The primary authorization ID that is passed to DB2 during sign-on. CICS ignores any changes made to this field by the exit routine. |
| CPRMUSER | 10 | Character, 4 bytes | Reserved for use by the exit routine. CICS preserves this field from one exit to the next. |

The field CPRMUSER can be used for such purposes as addressing a user table or even a CICS GETMAIN area. There is a unique field called CPRMUSER for each RCT entry with PLNEXIT=YES.

The following sample macros in *prefix*.SDSNMACS map the parameter list in the languages shown:

| | |
|---|---|
| **DSNCPRMA** | Assembler |
| **DSNCPRMC** | COBOL |
| **DSNCPRMP** | PL/I |

# General considerations for writing exit routines

The rules, requirements, and suggestions below apply to most of the foregoing exit routines.

**Attention:** Using an exit routine requires coordination with your system programmers. An exit routine runs as an extension of DB2 and has all the privileges of DB2. It can impact the security and integrity of the database. Conceivably, an exit routine could also expose the integrity of the operating system. Instructions for avoiding that exposure can be found in the appropriate MVS/ESA or OS/390 publication.

## Coding rules

An exit routine must conform to these rules:

- It must be written in assembler.
- It must reside in an authorized program library, either the library containing DB2 modules (*prefix*.SDSNLOAD) or in a library concatenated ahead of *prefix*.SDSNLOAD in the procedure for the database services started task (the procedure named *ssnm*DBM1, where *ssnm* is the DB2 subsystem name). Authorization routines must be accessible to the *ssnm*MSTR procedure. For all routines, we recommend using the library *prefix*.SDSNEXIT, which is concatenated ahead of *prefix*.SDSNLOAD in both started-task procedures.
- Routines listed below *must* have the names shown. The name of other routines should not start with “DSN”, to avoid conflict with the DB2 modules.

| Type of routine | Required load module name |
|---|---|
| **Date** | DSNXVDTX |
| **Time** | DSNXVTMX |
| **Connection** | DSN3@ATH |
| **Sign-on** | DSN3@SGN |

- It must be written to be reentrant and must restore registers before return.
- It must be link-edited with the REENTRANT parameter.
- In the MVS/ESA environment, it must be written and link-edited to execute AMODE(31),RMODE(ANY).
- It must not invoke any DB2 services—for example, through SQL statements.
- It must not invoke any SVC services or ESTAE routines.

Even though DB2 has functional recovery routines of its own, you can establish your own functional recovery routine (FRR), specifying MODE=FULLXM and EUT=YES.

## Modifying exit routines

Since exit routines operate as extensions of DB2, they should not be changed or modified while DB2 is running.

## Execution environment

Exit routines are invoked by standard CALL statements. With some exceptions, which are noted under “General Considerations” in the description of particular types of routine, the execution environment is:

- Supervisor state
- Enabled for interrupts

- PSW key 7
- No MVS locks held
- For local requests, under the TCB of the application program that requested the DB2 connection
- For remote requests, under a TCB within the DB2 distributed data facility address space
- 31-bit addressing mode
- Cross-memory mode

  In cross-memory mode, the current primary address space is not equal to the home address space. Hence, some MVS macro services you cannot use at all, and some you can use only with restrictions. For more information about cross-memory restrictions for macro instructions, which macros can be used fully, and the complete description of each macro, refer to the appropriate MVS/ESA or OS/390 publication.

## Registers at invocation

When DB2 passes control to an exit routine, the registers are set as follows:

| Register | Contains |
|---|---|
| **1** | Address of pointer to the exit parameter list (shown in Table 160). *For a field procedure*, the address is that of the field procedure parameter list (see Figure 136 on page 937). |
| **13** | Address of the register save area. |
| **14** | Return address. |
| **15** | Address of entry point of exit routine. |

## Parameter lists

Register 1 points to the address of parameter list EXPL, described by macro DSNDEXPL and shown in Figure 137. The word following points to a second parameter list, which differs for each type of exit routine.

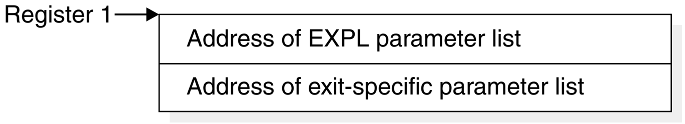

Register 1
Address of EXPL parameter list
Address of exit-specific parameter list

*Figure 137. Use of register 1 on invoking an exit routine. (Field procedures and translate procedures do not use the standard exit-specific parameter list.)*

The EXPL parameter list is shown below; its description is given by macro DSNDEXPL.

*Table 160. Contents of EXPL parameter list*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| EXPLWA | 0 | Address | Address of a work area to be used by the routine |
| EXPLWL | 4 | Signed 4-byte integer | Length of the work area. The value is:<br>2048 for connection and sign-on routines<br>512 for date and time routines and translate procedures (see Note 1).<br>256 for edit, validation, and log capture routines |

*Table 160. Contents of EXPL parameter list (continued)*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| EXPLRSV1 | 8 | Signed 2-byte integer | Reserved |
| EXPLRC1 | A | Signed 2-byte integer | Return code |
| EXPLRC2 | C | Signed 4-byte integer | Reason code |
| EXPLARC | 10 | Signed 4-byte integer | Used only by connection and sign-on routines |
| EXPLSSNM | 14 | Character, 8 bytes | Used only by connection and sign-on routines |
| EXPLCONN | 1C | Character, 8 bytes | Used only by connection and sign-on routines |
| EXPLTYPE | 24 | Character, 8 bytes | Used only by connection and sign-on routines |

**Notes:**

1. When translating a string of type PC MIXED, a translation procedure has a work area of 256 bytes plus the length attribute of the string.

# Row formats for edit and validation routines

In writing an edit or validation routine, you must be aware of the format in which DB2 stores the rows of tables. This section describes the special features of that format.

## Column boundaries

DB2 stores columns contiguously, regardless of word boundaries in physical storage.

LOB columns are an exception. LOB values are not stored contiguously. An indicator column is stored in a base table in place of the LOB value.

Edit procedures cannot be specified for any table that contains a LOB column or a ROWID column. In addition, LOB values are not available to validation routines; indicator columns and ROWID columns represent LOB columns as input to a validation procedure.

## Null values

If null values are allowed for a column, an extra byte is stored before the actual column value. This byte is X'00' if the column value is **not** null; it is X'FF' if the value is null.

The extra byte is included in the column length attribute (parameter FFMTFLEN in Table 162 on page 954).

## Fixed-length rows

If all columns in a table are fixed-length, its rows are stored in fixed-length format. The rows are merely byte strings.

For example, the sample project activity table has five fixed-length columns. The first two columns do not allow nulls; the last three do. Here is how a row in the table looks:

| Column 1 | Column 2 | Column 3 | | Column 4 | | Column 5 | |
|---|---|---|---|---|---|---|---|
| MA2100 | 10 | 00 | 0.5 | 00 | 820101 | 00 | 821101 |

## Varying-length rows

If a table has any varying-length columns, its rows contain varying-length values, and are varying-length rows. Each varying-length value has a 2-byte length field in front of it. Those 2 bytes are **not** included in the column length attribute (FFMTFLEN).

Here is how a row of the sample department table looks:

| Column 1 | Column 2 | | Column 3 | Column 4 |
|---|---|---|---|---|
| C01 | 0012 | Information Center | 000030 | A00 |

Column length (hex)

There are no gaps after varying-length columns. Hence, columns that appear after varying-length columns are at variable offsets in the row. To get to such a column, you must scan the columns sequentially after the first varying-length column. An empty string has a length of zero with no data following.

ROWID and indicator columns are treated like varying length columns. Row IDs are VARCHAR(17). An indicator columns is VARCHAR(4); it is stored in a base table in place of a LOB column, and indicates whether the LOB value for the column is null or zero length.

## Varying-length rows with nulls

A varying-length column can also allow null values. The value in the length field includes the null indicator byte but does not include the length field itself.

Here is how the same row would look in storage if nulls were allowed in the DEPTNAME column:

| Column 1 | Column 2 | | Column 3 | Column 4 |
|---|---|---|---|---|
| C01 | 0013 | Information Center | 000030 | A00 |

Column length (hex)

An empty string has a length of one, a X'00' null indicator, and no data following.

## Internal formats for dates, times, and timestamps

The values in columns with data types of DATE, TIME, and TIMESTAMP are stored in the formats shown in the following figure. In each format, each byte consists of two packed decimal digits.

DATE format: 4 bytes
Content

| Year | Month | Day |
|---|---|---|
| 2 | 1 | 1 |

Number of bytes

TIME format: 3 bytes
Content

| Hours | Minutes | Seconds |
|---|---|---|
| 1 | 1 | 1 |

Number of bytes

TIMESTAMP format: 10 bytes
Content

| Year | Month | Day | Hours | Minutes | Seconds | Microseconds |
|---|---|---|---|---|---|---|
| 2 | 1 | 1 | 1 | 1 | 1 | 3 |

Number of bytes

## Parameter list for row format descriptions

DB2 passes a description of the row format to an edit or validation routine through a parameter list, generated by macro DSNDROW. The description includes both the general row characteristics and the characteristics of each column. Table 161 shows the general row description, and Table 162 shows the description of each column.

*Table 161. Description of a row format*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| RFMTNFLD | 0 | Signed fullword integer | Number of columns in a row |
| RFMTAFLD | 4 | Address | Address of a list of column descriptions. The format of each column is shown in Table 162. |
| RFMTTYPE | 8 | Character, 1 byte | Row type:<br>X'00' = row with fixed-length columns<br>X'04' = row with varying-length columns |
|  | 9 | Character, 3 bytes | Reserved |

*Table 162. Description of a column format*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| FFMTFLEN | 0 | Signed fullword integer | Column length attribute (see Table 163 on page 955) |
| FFMTFTYP | 4 | Character, 1 byte | Data type code (see Table 163 on page 955) |
| FFMTNULL | 5 | Character, 1 byte | Data attribute:<br>X'00' = Null values are allowed.<br>X'04' = Null values are not allowed. |
| FFMTFNAM | 6 | Character, 18 bytes | Column name |

*Table 163. Description of data type codes and length attributes*

| Data type | Code (FFMTFTYP) | Length attribute (FFMTFLEN) |
|---|---|---|
| INTEGER | X'00' | 4 |
| SMALLINT | X'04' | 2 |
| FLOAT (single precision) | X'08' | 4 |
| FLOAT (double precision) | X'08' | 8 |
| DECIMAL | X'0C' | INTEGER(*p*/2), where *p* is the precision |
| CHAR | X'10' | The length of the string |
| VARCHAR | X'14' | The length of the string |
| DATE | X'20' | 4 |
| TIME | X'24' | 3 |
| TIMESTAMP | X'28' | 10 |
| ROWID | X'2C' | 17 |
| INDICATOR COLUMN | X'30' | 4 |

## DB2 codes for numeric data

DB2 stores numeric data in a specially encoded format. That format is called *DB2-coded*. To retrieve numeric data in its original form, you must *DB2-decode* it, according to its data type, as follows:

**Data type** **DB2 decoding procedure**

**SMALLINT**
Invert the sign bit (high order bit).

| Value | Means ... |
|---|---|
| **8001** | 0001 (+1 decimal) |
| **7FF3** | FFF3 (-13 decimal) |

**INTEGER**
Invert the sign bit (high order bit).

| Value | Means ... |
|---|---|
| **800001F2** | 000001F2 (+498 decimal) |
| **7FFFFF85** | FFFFFF85 (-123 decimal) |

**FLOAT**
If the sign bit (high order bit) is 1, invert only that bit. Otherwise, invert all bits.

| Value | Means ... |
|---|---|
| **C110000000000000** | 4110000000000000 (+1.0 decimal) |
| **3EEFFFFFFFFFFFFF** | C110000000000000 (-1.0 decimal) |

**DECIMAL**
Save the high-order hexadecimal digit (sign digit). Shift the number to the left one hexadecimal digit. If the sign digit is X'F', put X'C' in the low-order position. Otherwise, invert all bits in the number and put X'D' in the low-order position.

| Value | Means ... |
|---|---|
| **F001** | 001C (+1) |
| **0FFE** | 001D (-1) |

## Routine for CICS transaction invocation stored procedure

The DB2–supplied CICS transaction routine stored procedure invokes a user exit that you use to change values that the stored procedure caller provides. For information about this stored procedure, see “The CICS transaction invocation stored procedure (DSNACICS)” on page 1087.

#

# Appendix C. Reading log records

The information in this appendix is Product-sensitive Programming Interface and Associated Guidance Information as defined in “Notices” on page 1095.

This appendix discusses the following information about the log:
“What the log contains”
“The physical structure of the log” on page 962

For diagnostic or recovery purposes, it can be useful to read DB2 log records. This appendix also discusses three approaches to writing programs that read log records:

- “Reading log records with IFI” on page 968

  This is an online method using the instrumentation facility interface (IFI) when DB2 is running. You use the READA (read asynchronously) command of IFI to read log records into a buffer and the READS (read synchronously) command to pick up specific log control intervals from a buffer.
- “Reading log records with OPEN, GET, and CLOSE” on page 971

  This is a stand-alone method that can be used when DB2 is down. You use the assembler language macro DSNJSLR to submit OPEN, GET, and CLOSE functions. This method can be used to capture log records that you cannot pick up with IFI after DB2 goes down.
- “Reading log records with the log capture exit” on page 980

  This is an online method using the log capture exit when DB2 is running. You write an exit routine to use this exit to capture and transfer log records in real time.

## What the log contains

The log contains the information needed to recover the results of program execution, the contents of the database, and the DB2 subsystem. It does not contain information for accounting, statistics, traces, or performance evaluation.

There are three main types of log records which are described under these headings:
“Unit of recovery log records” on page 958
“Checkpoint log records” on page 961
“Database page set control records” on page 962

Exception information that is not included in any of these types is described under “Other exception information” on page 962.

Each log record has a header that tells its type, the DB2 subcomponent that made the record, and, for unit of recovery records, the unit of recovery identifier. The log records can be extracted and printed by the DSN1LOGP program. For instructions, refer to Part 3 of *DB2 Utility Guide and Reference*.

***The log relative byte address and log record sequence number:*** The DB2 log can contain up to $2^{48}$ bytes, where $2^{48}$ is 2 to the 48th power. Each byte is addressable by its offset from the beginning of the log. That offset is known as its *relative byte address* (RBA).

A log record is identifiable by the RBA of the first byte of its header; that RBA is called the *relative byte address of the record*. The record RBA is like a timestamp because it uniquely identifies a record that starts at a particular point in the continuing log.

In the data sharing environment, each member has its own log. A means is therefore needed to identify log records uniquely across the data sharing group. The *log record sequence number (LRSN)* provides that means. The LRSN is a 6-byte hexadecimal value derived from a store clock timestamp. DB2 uses the LRSN for recovery in the data sharing environment.

***Effects of ESA data compression:*** Log records can contain compressed data if a table contains compressed data. For example, if the data in a DB2 row are compressed, all data logged because of changes to that row (resulting from inserts, updates and deletes) are compressed. If logged, the record prefix is not compressed, but all of the data in the record are in compressed format. Reading compressed data requires access to the dictionary that was in use when the data was compressed.

## Unit of recovery log records

Most of the log records describe changes to the DB2 database. All such changes are made within units of recovery. We first describe those, and their effects, and then the corresponding log records.

### Undo and redo records

When a change is made to the database, DB2 logs an *undo/redo* record that describes the change. The *redo* information is required if the work is committed and later must be recovered. The *undo* information is used to back out work that is not committed.

If the work is rolled back, the undo/redo record is used to remove the change. At the same time that the change is removed, a new redo/undo record is created that contains information, called *compensation information*, that is used if necessary to reverse the change. For example, if a value of 3 is changed to 5, redo compensation information changes it back to 3.

If the work must be recovered, DB2 scans the log forward and applies the redo portions of log records and the redo portions of compensation records, without keeping track of whether the unit of recovery was committed or rolled back. If the unit of recovery had been rolled back, DB2 would have written compensation redo log records to record the original undo action as a redo action. Using this technique, the data can be completely restored by applying only redo log records on a single forward pass of the log.

DB2 also logs the creation and deletion of data sets. If the work is rolled back, the operations are reversed. For example, if a table space is created using DB2-managed data sets, DB2 creates a data set; if rollback is necessary, the data set is deleted. If a table space using DB2-managed data sets is dropped, DB2 deletes the data set when the work is committed, not immediately. If the work is rolled back, DB2 does nothing.

### Database exception table records

Database exception table (DBET) log records register several types of information: exception states and image copies of special table spaces. DBET log records also register exception information that is not related to units of recovery (see “Other exception information” on page 962 for more information.

***Exception states:*** DBET log records register whether any database, table space, index space, or partition is in an exception state. To list all objects in a database that are in an exception state, use the command DISPLAY DATABASE (*database name*) RESTRICT. For a further explanation of the list produced and of the exception states, see the description of message DSNT392I in Part 2 of *DB2 Messages and Codes*.

***Image copies of special table spaces:*** Image copies of DSNDB01.SYSUTILX, DSNDB01.DBD01, and DSNDB06.SYSCOPY are registered in the DBET log record rather than in SYSCOPY. During recovery, they are recovered from the log, and then image copies of other table spaces are located from the recovered SYSCOPY.

## Typical unit of recovery log records

Table 164 shows a sequence of log records that might be written for an insert of one row via TSO. The following record types are included:

**Begin_UR**
The first request to change a database begins a unit of recovery. The log record of that event is identified by its log RBA. That same RBA serves as an ID for the entire unit of recovery (the URID). All records related to that unit have that RBA in their log record headers (LRH). For rapid backout, the records are also linked by a backward chain in the LRH.

**Undo/Redo**
Log records are written for each insertion, deletion, or update of a row. They register the changes to the stored data, but not the SQL statement that caused the change. Each record identifies one data or index page and its changes.

**End Phase 2 records**
The end of a UR is marked by log records that tell whether the UR was committed or rolled back, and whether DB2 has completed the work associated with it. If DB2 terminates before a UR has completed, it completes the work at the next restart.

*Table 164. Example of a log record sequence for an INSERT of one row using TSO*

| Type of record | Information recorded |
|---|---|
| 1. Begin_UR | Beginning of the unit of recovery. Includes the connection name, correlation name, authorization ID, plan name, and LUWID. |
| 2. Undo/Redo for data | Insertion of data. Includes the database ID (DBID), page set ID, page number, internal record identifier (RID), and the data inserted. |
| 3. Undo/Redo for Index | Insertion of index entry. Includes the DBID, index space object ID, page number, and index entry to be added. |
| 4. Begin Commit 1 | The beginning of the commit process. The application has requested a commit either explicitly (EXEC SQL COMMIT) or implicitly (for example, by ending the program). |
| 5. Phase 1-2 Transition | The agreement to commit in TSO. In CICS and IMS, an End Phase 1 record notes that DB2 agrees to commit. If both parties agree, a Begin Phase 2 record is written; otherwise, a Begin Abort record is written, noting that the unit of recovery is to be rolled back. |
| 6. End Phase 2 | Completion of all work required for commit. |

Table 165 shows the log records for processing and rolling back an insertion.

*Table 165. Log records written for rolling back an insertion*

| Type of record | Information recorded |
|---|---|
| 1. Begin_UR | Beginning of the unit of recovery. |
| 2. Undo/Redo for data | Insertion of data. Includes the database ID (DBID), page set ID, page number, internal record identifier, and the data inserted. |
| 3. Begin_Abort | Beginning of the rollback process. |
| 4. Compensation Redo/Undo | Backing-out of data. Includes the database ID (DBID), page set ID, page number, internal record ID (RID), and data to undo the previous change. |
| 5. End_Abort | End of the unit of recovery, with rollback complete. |

## Classes of changes to data

Table 166 summarizes the information logged for data and index changes.

*Table 166. Information logged for database changes*

| Operation | Information logged |
|---|---|
| Insert data | The new row<br>• On redo, the row is inserted with its original RID.<br>• On undo, the row is deleted and the RID is made available for another row. |
| Delete data | The deleted row<br>• On redo, the RID is made available for another row.<br>• On undo, the row is inserted again with its former RID. |
| Update data | The old and new values of the changed data.<br>• On redo, the new data is replaced.<br>• On undo, the old data is replaced. |
| **Note:** If an update occurs to a table defined with DATA CAPTURE(CHANGES), the entire before-image and after-image of the data row is logged. | |
| Insert index entry | The new key value and the data RID. |
| Delete index entry | The deleted key value and the data RID. |

There are three basic classes of changes to a data page:

- Changes to control information. Those changes include pages that map available space and indicators that show that a page has been modified. The COPY utility uses that information when making incremental image copies.
- Changes to database pointers. Pointers are used in two situations:
  - The DB2 catalog and directory, but not user databases, contain pointers that connect related rows. Insertion or deletion of a row changes pointers in related data rows.
  - When a row in a user database becomes too long to fit in the available space, it is moved to a new page. An address, called an *overflow pointer*, that points to the new location is left in the original page. With this technique, index entries and other pointers do not have to be changed. Accessing the row in its original position gives a pointer to the new location.
- Changes to data. In DB2, a row is confined to a single page. Each row is uniquely identified by a RID containing:
  - The number of the page

- A 1-byte ID that identifies the row within the page. A single page can contain up to 255 rows; [12] IDs are reused when rows are deleted.

The log record identifies the RID, the operation (insert, delete, or update), and the data. Depending on the data size and other variables, DB2 can write a single log record with both undo and redo information, or it can write separate log records for undo and redo.

## Checkpoint log records

To reduce restart time, DB2 takes periodic checkpoints during normal operation, in the following circumstances:

- When a predefined number of log records have been written

  This number is defined by field CHECKPOINT FREQ on installation panel DSNTIPN described in Part 2 of *DB2 Installation Guide*.
- When switching from one active log data set to another
- At the end of a successful restart
- At normal termination

At a checkpoint, DB2 logs its current status and registers the log RBA of the checkpoint in the bootstrap data set (BSDS). At restart, DB2 uses the information in the checkpoint records to reconstruct its state when it terminated.

Many log records can be written for a single checkpoint. DB2 can write one to begin the checkpoint; others can then be written, followed by a record to end the checkpoint. Table 167 summarizes the information logged.

*Table 167. Contents of checkpoint log records*

| Type of log record | Information logged |
|---|---|
| Begin_Checkpoint | Marks the start of the summary information. All later records in the checkpoint have type X'0100' (in the LRH). |
| Unit of Recovery Summary | Identifies an incomplete unit of recovery (by the log RBA of the Begin_UR log record). Includes the date and time of its creation, its connection ID, correlation ID, authorization ID, the plan name it used, and its current state (inflight, indoubt, in-commit, or in-abort). |
| Page Set Summary | Contains information for allocating and opening objects at restart, and identifies (by the log RBA) the earliest checkpoint interval containing log records about data changes that have not been applied to the DASD version of the data or index. There is one record for each open page set (table space or index space). |
| Page Set Exception Summary | Identifies the type of exception state. For descriptions of the states, see “Database page set control records” on page 962 below. There is one record for each database and page set with an exception state. |
| Page Set UR Summary Record | Identifies page sets modified by any active UR (inflight, in-abort, or in-commit) at the time of the checkpoint. |
| End_Checkpoint | Marks the end of the summary information about a checkpoint. |

[12] A page in a catalog table space that has links can contain up to 127 rows.

## Database page set control records

Page set control records primarily register the allocation, opening, and closing of every page set (table space or index space). That same information is in the DB2 directory (SYSIBM.SYSLGRNX). It is also registered in the log so that it is available at restart.

## Other exception information

Entries for data pages that are logically in error (logical page list or LPL entries) or physically in error (write error page range or WEPR entries) are registered in the DBET log record.

# The physical structure of the log

Each active log data set must be a VSAM linear data set (LDS). The physical output unit written to the active log data set is a control interval (CI) of 4096 bytes (4 KB). Each CI contains one VSAM record.

## Physical and logical log records

The VSAM CI provides 4089 bytes to hold DB2 information. We refer to that space as a *physical* record. The information to be logged at a particular time forms a *logical* record, whose length varies independently of the space available in the CI. Hence, one physical record can contain several logical records, one or more logical records and part of another, or only part of one logical record. The physical record must also contain 21 bytes of DB2 control information, called the *log control interval definition* (LCID), which is further described in “The log control interval definition (LCID)” on page 964.

Figure 138 on page 963 shows a VSAM CI containing four log records or segments, namely:

- The last segment of a log record of 768 bytes (X'0300'). The length of the segment is 100 bytes (X'0064').
- A complete log record of 40 bytes (X'0028').
- A complete log record of 1024 bytes (X'0400').
- The first segment of a log record of 4108 bytes (X'100C'). The length of the segment is 2911 bytes (X'0B5F').

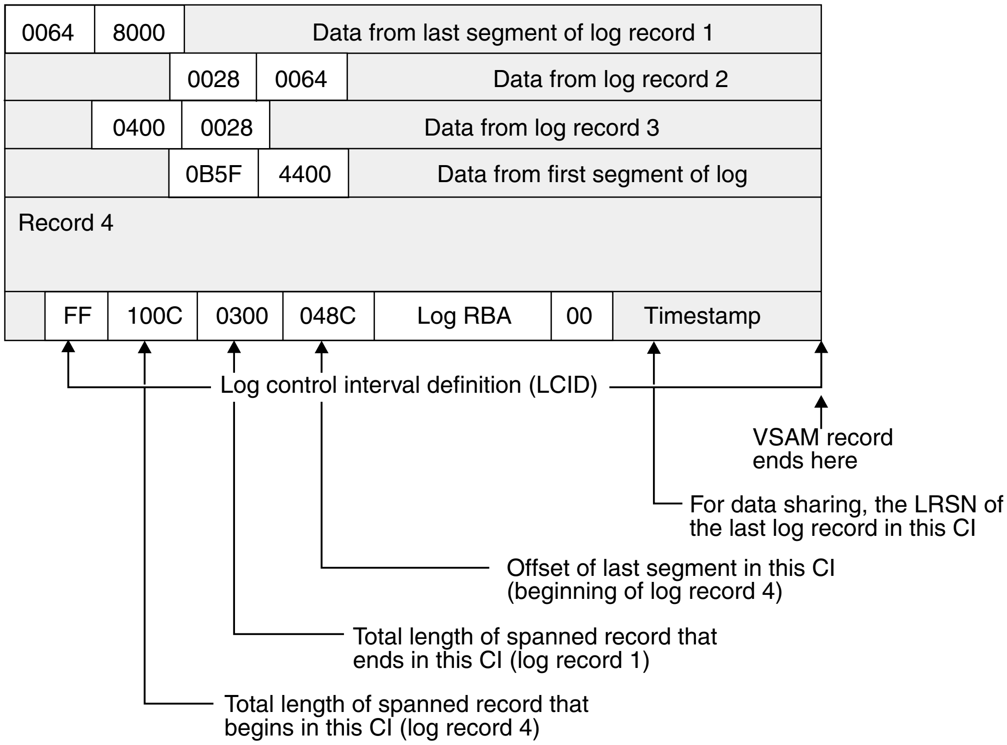

0064
8000
Data from last segment of log record 1
0028
0064
Data from log record 2
0400
0028
Data from log record 3
0B5F
4400
Data from first segment of log
Record 4
FF
100C
0300
048C
Log RBA
00
Timestamp
Log control interval definition (LCID)
VSAM record ends here
For data sharing, the LRSN of the last log record in this CI
Offset of last segment in this CI (beginning of log record 4)
Total length of spanned record that ends in this CI (log record 1)
Total length of spanned record that begins in this CI (log record 4)

*Figure 138. A VSAM CI and its contents*

We use the term *log record* to refer to a logical record, unless the term *physical log record* is used. A part of a logical record that falls within one physical record is called a *segment*.

## The log record header

Each logical record includes a prefix, called a *log record header* (LRH), which contains control information. For the contents of the log record header see Table 168.

The first segment of a log record must contain the header and some bytes of data. If the current physical record has too little room for the minimum segment of a new record, the remainder of the physical record is unused, and a new log record is written in a new physical record.

The log record can span many VSAM CIs. For example, a minimum of nine CIs are required to hold the maximum size log record of 32815 bytes. Only the first segment of the record contains the entire LRH; later segments include only the first two fields. When a specific log record is needed for recovery, all segments are retrieved and presented together as if the record were stored continuously.

*Table 168. Contents of the log record header*

| Hex offset | Length | Information |
|---|---|---|
| 00 | 2 | Length of this record or segment |

*Table 168. Contents of the log record header (continued)*

| Hex offset | Length | Information |
|---|---|---|
| 02 | 2 | Length of any previous record or segment in this CI; 0 if this is the first entry in the CI. The two high-order bits tell the segment type:<br>**B'00'** A complete log record<br>**B'01'** The first segment<br>**B'11'** A middle segment<br>**B'10'** The last segment |
| 04 | 2 | Type of log record [1] |
| 06 | 2 | Subtype of the log record [1] |
| 08 | 1 | Resource manager ID (RMID) of the DB2 component that created the log record |
| 09 | 1 | Flags |
| 0A | 6 | Unit of recovery ID, if this record relates to a unit of recovery[2]; otherwise, 0 |
| 10 | 6 | Log RBA of the previous log record, if this record relates to a unit of recovery[2]; otherwise, 0 |
| 16 | 1 | Release identifier |
| 17 | 1 | Length of header |
| 18 | 6 | Undo next LSN |
| 1E | 8 | LRHTIME |

**Note:**

[1] For record types and subtypes, see "Log record type codes" on page 966 and "Log record subtype codes" on page 966.

[2] For a description of units of recovery, see "Unit of recovery log records" on page 958.

## The log control interval definition (LCID)

Each physical log record includes a suffix called the *log control interval definition* (LCID), which tells how record segments are placed in the physical control interval. For the contents of the LCID, see Table 169.

*Table 169. Contents of the log control interval definition*

| Hex offset | Length | Information |
|---|---|---|
| 00 | 1 | Whether the CI contains free space: X'00' = Yes, X'FF' = No |
| 01 | 2 | Total length of a segmented record that begins in this CI; 0 if no segmented record begins in this CI |
| 03 | 2 | Total length of a segmented record that ends in this CI; 0 if no segmented record ends in this CI |
| 05 | 2 | Offset of the last record or segment in the CI |
| 07 | 6 | Log RBA of the start of the CI |
| # 0D | 6 | Timestamp, reflecting the date and time the log buffer was written to the active log data set. The timestamp is the high order six bytes of the Store Clock value (STCK). |
| # 13 | 2 | Reserved |

Each recovery log record consists of two parts: a *header*, which describes the record, and *data*. Figure 139 shows the format schematically; the list below it describes each field.

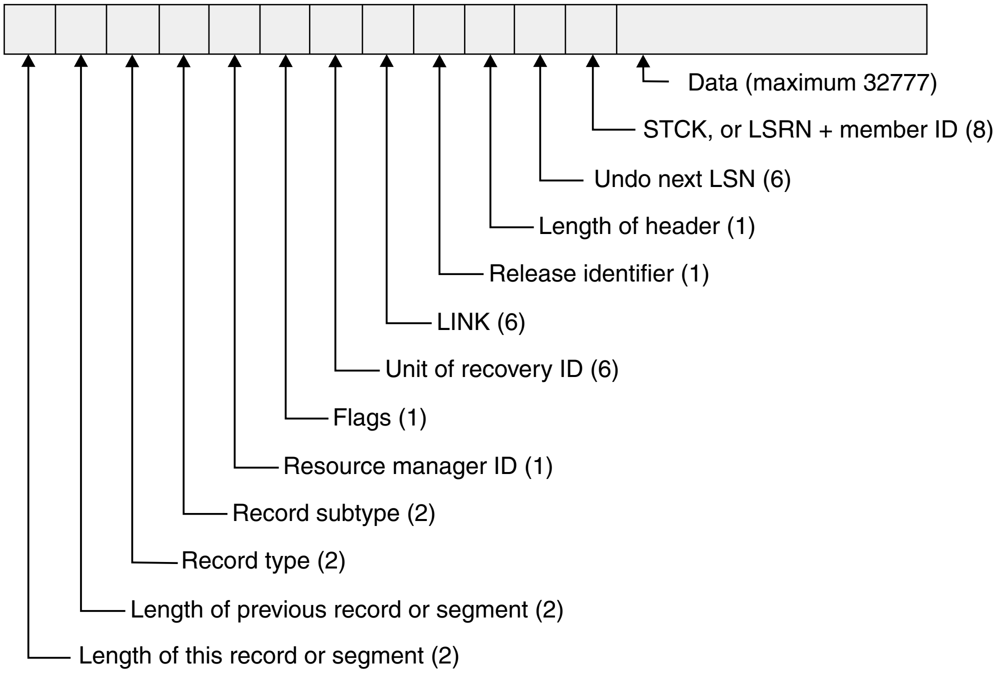


*Figure 139. Format of a DB2 recovery log record*

The fields are:

**Field** **Description**

**Length of this record**
The total length of the record in bytes.

**Length of previous record**
The total length of the previous record in bytes.

**Type** The code for the type of recovery log record. See “Log record type codes” on page 966.

**Subtype**
Some types of recovery log records are further divided into subtypes. See “Log record subtype codes” on page 966.

**Resource manager ID**
Identifier of the resource manager that wrote the record into the log. When the log is read, the record can be given for processing to the resource manager that created it.

**Unit of recovery ID**
A unit of recovery to which the record is related. Other log records can be related to the same unit of recovery; all of them must be examined to recover the data. The URID is the RBA (relative byte address) of the Begin-UR log record, and indicates the start of that unit of recovery in the log.

**LINK** Chains all records written using their RBAs. For example, the link in an end checkpoint record links the chains back to the begin checkpoint record.

**Release identifier**
Identifies in which release the log was written.

**Log record header length**
The total length of the header of the log record.

**Undo next LSN**
Identifies the log RBA of the next log record to be undone during backwards (UNDO processing) recovery.

**STCK, or LRSN+member ID.**
In a non data-sharing environment, this is a 6-byte store clock value (STCK) reflecting the date and time the record was placed in the output buffer. The last two bytes contain zeros.

In a data sharing environment, this contains a 6-byte log record sequence number (LRSN) followed by a 2-byte member ID.

**Data** Data associated with the log record. The contents of the data field depend on the type and subtype of the recovery log record.

## Log record type codes

The type code of a log record tells what kind of DB2 event the record describes:

| Code | Type of Event |
|---|---|
| **0002** | Page set control |
| **0004** | SYSCOPY utility |
| **0010** | System event |
| **0020** | Unit of recovery control |
| **0100** | Checkpoint |
| **0200** | Unit of recovery undo |
| **0400** | Unit of recovery redo |
| **0800** | Archive log command |
| **1000 to 8000** | Assigned by DB2 |
| **2200** | Savepoint |

A single record can contain multiple type codes that are combined. For example, 0600 is a combined UNDO/REDO record; F400 is a combination of four DB2-assigned types plus a REDO.

## Log record subtype codes

The log record subtype code provides a more granular definition of the event that occurred to produce the log record. Log record subtype codes are unique only within the scope of the corresponding log record type code.

Log record type 0004 (SYSCOPY utility) has log subtype codes that correspond to the page set ID values of the table spaces that have their SYSCOPY records in the log (SYSIBM.SYSUTILX, SYSIBM.SYSCOPY and DSNDB01.DBD01).

For a description of log record types 0200 (unit of recovery undo) and 0400 (unit of recovery redo), see the SUBTYPE option of DSN1LOGP in Part 3 of *DB2 Utility Guide and Reference*.

Log record type 0800 (quiesce) does not have subtype codes.

Some log record types (1000 to 8000 assigned by DB2) can have proprietary log record subtype codes assigned.

***Subtypes for type 0002 (page set control):***

| Code | Type of Event |
|---|---|
| **0001** | Page set open |
| **0002** | Data set open |
| **0003** | Page set close |
| **0004** | Data set close |
| **0005** | Page set control checkpoint |
| **0006** | Page set write |
| **0007** | Page set write I/O |
| **0008** | Page set reset write |
| **0009** | Page set status |

***Subtypes for type 0010 (system event):***

| Code | Type of Event |
|---|---|
| **0001** | Begin checkpoint |
| **0002** | End checkpoint |
| **0003** | Begin current status rebuild |
| **0004** | Begin historic status rebuild |
| **0005** | Begin active unit of recovery backout |
| **0006** | Pacing record |

***Subtypes for type 0020 (unit of recovery control):***

| Code | Type of Event |
|---|---|
| **0001** | Begin unit of recovery |
| **0002** | Begin commit phase 1 (Prepare) |
| **0004** | End commit phase 1 (Prepare) |
| **0008** | Begin commit phase 2 |
| **000C** | Commit phase 1 to commit phase 2 transition |
| **0010** | End commit phase 2 |
| **0020** | Begin abort |
| **0040** | End abort |
| **0081** | End undo |
| **0084** | End todo |
| **0088** | End redo |

***Subtypes for type 0100 (checkpoint):***

| Code | Type of Event |
|---|---|
| **0001** | Unit of recovery entry |
| **0002** | Restart unit of recovery entry |

***Subtypes for type 2200 (savepoint):***

| Code | Type of Event |
|---|---|
| **0014** | Rollback to savepoint |
| **000E** | Release to savepoint |

## Interpreting data change log records

DB2 provides the mapping and description of specific log record types that allow you to interpret data changes made to DB2 tables from the log. The macros are contained in the data set library *prefix*.SDSNMACS and are documented by comments in the macros themselves.

Log record formats for the record types and subtypes listed above are detailed in the mapping macro DSNDQJ00. DSNDQJ00 provides the mapping of specific data change log records, UR control log records, and page set control log records that you need to interpret data changes by the UR. DSNDQJ00 also explains the content and usage of the log records.

## Reading log records with IFI

You can write a program that uses IFI to capture log records while DB2 is running. You can read the records asynchronously, by starting a trace that reads the log records into a buffer and then issuing an IFI call to read those records out of the buffer. Or, you can read those log records synchronously, by using an IFI call that returns those log records directly to your IFI program.

This section describes both methods, in the following topics:
“Reading log records into a buffer”
“Reading specific log records (IFCID 0129)”
“Reading complete log data (IFCID 0306)” on page 969

Either the primary or one of the secondary authorization IDs must have MONITOR2 privilege. For details on how to code an IFI program, see “Appendix E. Programming for the Instrumentation Facility Interface (IFI)” on page 997.

### Reading log records into a buffer

To begin gathering active log records into a buffer, issue the following command in your IFI program:

```
-START TRACE(P) CLASS(30) IFCID(126) DEST(OPX)
```

Where:

- P signifies to start a DB2 performance trace. Any of the DB2 trace types can be used.
- CLASS(30) is a user-defined trace class (31 and 32 are also user-defined classes).
- IFCID(126) activates DB2 log buffer recording.
- DEST(OPX) starts the trace to the next available DB2 online performance (OP) buffer. The size of this OP buffer can be explicitly controlled by the BUFSIZE keyword of the START TRACE command. Valid sizes range from 8 KB to 1 MB in 4 KB increments.

When the START TRACE command takes effect, from that point forward, until DB2 terminates, DB2 will begin writing 4 KB log buffer VSAM control intervals (CIs) to the OP buffer as well as to the active log. As part of the IFI COMMAND invocation, the application specifies an ECB to be posted and a threshold to which the OP buffer is filled when the application is posted to obtain the contents of the buffer. The IFI READA request is issued to obtain OP buffer contents.

### Reading specific log records (IFCID 0129)

IFCID 129 can be used with an IFI READS request to return a specific range of log records from the active log into the return area your program has initialized. Enter the following command into your IFI program:

```
CALL DSNWLI(READS,ifca,return_area,ifcid_area,qual_area)
```

IFCID 129 must appear in the IFCID area.

To retrieve the log control interval, your program must initialize certain fields in the qualification area:

**WQALLTYP**
This is a 3-byte field in which you must specify CI (with a trailing blank), which stands for “control interval”.

**WQALLMOD**
In this 1-byte field, you specify whether you want the first log CI of the restarted DB2 subsystem, or whether you want a specific control interval as specified by the value in the RBA field.

**F** The “first” option is used to retrieve the first log CI of this DB2 instance. This option ignores any value in WQALLRBA and WQALLNUM.

**P** The“partial” option is used to retrieve partial log CIs for the log capture exit which is described in Appendix B. DB2 places a value in field IFCAHLRS of the IFI communication area, as follows:

- The RBA of the log CI given to the log capture exit, if the last CI written to the log was not full.
- 0, if the last CI written to the log was full.

When you specify option P, DB2 ignores values in WQALLRBA and WQALLNUM.

**R** The “read” option is used to retrieve a set of up to 7 continuous log CIs. If you choose this option, you must also specify the WQALLRBA and WQALLNUM options explained below.

**WQALLRBA**
In this 8-byte field, you specify the starting log RBA of the control intervals to be returned. This value must end in X'000' to put the address on a valid boundary. This field is ignored when using the WQALLMOD=F option.

If you specify an RBA that is not in the active log, reason code 00E60854 is returned in the field IFCARC2, and the RBA of the first CI of the active log is returned in field IFCAFCI of the IFCA. These 6 bytes contain the IFCAFCI field.

**WQALLNUM**
In this 2-byte field, specify the number of control intervals you want returned. The valid range is from X'0001' through X'0007', which means that you can request and receive up to seven 4 KB log control intervals. This field is ignored when using the WQALLMOD=F option. For a complete description of the qualification area, see Table 182 on page 1004.

If you specify a range of log CIs, but some of those records have not yet been written to the active log, DB2 returns as many log records as possible. You can find the number of CIs returned in field QWT02R1N of the self-defining section of the record. For information about interpreting trace output, see “Appendix D. Interpreting DB2 trace output” on page 981.

## Reading complete log data (IFCID 0306)

The major benefits for using IFCID 0306 are:

- IFCID 0306 can request DB2 to decompress log records if compressed, before passing them to the return area of your IFI program.
- In a data sharing environment, DB2 merges log records if the value of the IFI READS qualification WQALFLTR is X'00'. If WQALFLTR is X'01', log records are not merged.
- IFCID can retrieve log records from the archive data sets.
- Complete log records are always returned.

To use this IFCID, use the same call as described in “Reading specific log records (IFCID 0129)” on page 968. IFCID 0306 must appear in the IFCID area. IFCID 0306 returns complete log records and the spanned record indicators in bytes 2 will have no meaning, if present. Multi-segmented control interval log records are combined for a complete log record.

## Specifying the return area

For IFCID 0306 requests, your program’s return area must reside in ECSA key 7 storage. The IFI application program must set the eye-catcher to ’I306’ at offset 4 in the Return Area before making the IFCID 0306 call. There is an additional 60 byte area that must be included after the 4-byte length indicator and the ’I306’ eye-catcher. This area is used by DB2 between successive application calls and must not be modified by the application. The return area mapping is documented later in this section.

The IFI application program needs to run in supervisor state to request the ECSA key 7 return area. The return area storage size need be a minimum of the largest DB2 log record returned plus the additional area defined in DSNDQW04. Minimize the number of IFI calls required to get all log data but do not over use ECSA by the IFI program. The other IFI storage areas can remain in user storage key 8. The IFI application must be in supervisor state and key 0 when making IFCID 0306 calls.

## Qualifying log records

To retrieve IFCID 0306 log records, your program must initialize certain fields in the qualification area mapped by DSNDWQAL. These qualification fields are described here:

**WQALLMOD**

In this 1-byte field, specify one of the following modes:

**D** Retrieves the single log record whose RBA value and member id is specified in WQALLRBA. Issuing a D request while holding a position in the log, causes the request to fail and terminates the log position held.

**F** Used as a first call to request log records beyond the LRSN or RBA specified in WQALLRBA that meet the criteria specified in WQALLCRI.

**H** Retrieves the highest LRSN or log RBA in the active log. The value is returned in field IFCAHLRS of the IFI communications area (IFCA). There is no data returned in the return area and the return code for this call will indicate that no data was returned.

**N** Used following mode F or N calls to request any remaining log records that meet the criteria specified in WQALLCRI. * and any option specified in WQALLOPT. As many log records as fit in the program’s return area are returned.

**T** Terminates the log position that was held by any previous F or N request. This allows held resources to be released.

Mode R is not used for IFCID 0306.

For both F or N requests, each log record returned contains a record-level feedback area recorded in QW0306L. The number of log records retrieved is in QW0306CT. The ending log RBA or LRSN of the log records to be returned is in QW0306ES.

**WQALLRBA**

In this 8-byte field, specify the starting log RBA or LRSN of the control records to be returned. For IFCID 0306, this is used on the “first” option (F) request to request log records beyond the LRSN or RBA specified in this field. Determine the RBA or LRSN value from the H request. For RBAs, the value plus one should be used. For IFCID 0306 with D request of WQALLMOD, the high order 2 bytes must specify member id and the low order 6 bytes contain the RBA.

**WQALLCRI**
In this 1-byte field, indicate what types of log records you want:

**X'00'**
Tells DB2 to retrieve only log records for changed data capture and unit of recovery control.

**X'FF'**
Tells DB2 to retrieve all types of log records. Use of this option can retrieve large data volumes and degrade DB2 performance.

**WQALLOPT**
In this 1-byte field, indicate whether you want the returned log records to be decompressed.

**X'01'**
Tells DB2 to decompress the log records before they are returned.

**X'00'**
Tells DB2 to leave the log records in the compressed format.

***A typical sequence of IFCID 0306 calls is::***

**WQALLMOD=H**
This is only necessary if you want to find the current position in the log. The LRSN or RBA is returned in IFCAHLRS. The return area is not used.

**WQALLMOD=F**
The WQALLLRBA, WQALLLCRI and WQALLLOPT should be set. If 00E60812 is returned, you have all the data for this scope. You should wait a while before issuing another WQALLMOD=F call. In data sharing, log buffers are flushed when the F request is issued.

**WQALLMOD=N**
If the 00E60812 has not been returned, you issue this call until it is. You should wait a while before issuing another WQALLMOD=F call.

**WQALLMOD=T**
This should only be used if you do not want to continue with the WQALLMOD=N before the end is reached. It has no use if a position is not held in the log.

***IFCID 0306 return area mapping:*** IFCID 0306 has a unique return area format. The first section is mapped by QW0306OF instead of the writer header DSNDQWIN. See “Appendix E. Programming for the Instrumentation Facility Interface (IFI)” on page 997 for details.

## Reading log records with OPEN, GET, and CLOSE

DB2 provides three stand-alone log services that user-written application programs can use to read DB2 recovery log records and control intervals even when DB2 is not running:

- OPEN initializes stand-alone log services.
- GET returns a pointer to the next log record or log record control interval.
- CLOSE deallocates data sets and frees storage.

To invoke these services, use the assembler language macro, DSNJSLR, specifying one of the above functions.

These log services use a *request block*, which contains a feedback area in which information for all stand-alone log GET calls is returned. The request block is created when a stand-alone log OPEN call is made. The request block must be passed as input to all subsequent stand-alone log calls (GET and CLOSE). The request block is mapped by the DSNDSLRB macro and the feedback area is mapped by the DSNDSLRF macro.

See Figure 140 on page 979 for an example of an application program that includes these various stand-alone log calls.

When you issue an OPEN request, you can indicate whether you want to get log records or log record control intervals. Each GET request returns a single logical record or control interval depending on which you selected with the OPEN request. If neither is specified, the default, RECORD, is used. DB2 reads the log in the forward direction of ascending relative byte addresses or log record sequence numbers (LRSNs).

If a bootstrap data set (BSDS) is allocated before stand-alone services are invoked, appropriate log data sets are allocated dynamically by MVS. If the bootstrap data set is not allocated before stand-alone services are invoked, the JCL for your user-written application to read a log must specify and allocate the log data sets to be read.

Table 170 lists and describes the JCL DD statements used by stand-alone services.

*Table 170. JCL DD statements for DB2 stand-alone log services*

| **JCL DD statement** | **Explanation** |
|---|---|
| JOBCAT or STEPCAT | Specifies the catalog in which the BSDS and the active log data sets are cataloged. Required if the BSDS or any active log data set is to be accessed, unless the data sets are cataloged in the system master catalog. |
| BSDS | Specifies the bootstrap data set (BSDS). Optional. Another ddname can be used for allocating the BSDS, in which case the ddname must be specified as a parameter on the FUNC=OPEN (see “Stand-alone log OPEN request” on page 975 for more information). Using the ddname in this way causes the BSDS to be used. If the ddname is omitted on the FUNC=OPEN request, the processing uses DDNAME=BSDS when attempting to open the BSDS. |
| ARCHIVE | Specifies the archive log data sets to be read. Required if an archive data set is to be read and the BSDS is not available (the BSDS DD statement is omitted). Should not be present if the BSDS DD statement is present. If multiple data sets are to be read, specify them as concatenated data sets in ascending log RBA order. |
| ACTIVE*n* | (Where *n* is a number from 1 to 7). Specifies an active log data set that is to be read. Should not be present if the BSDS DD statement is present. If only one data set is to be read, use ACTIVE1 as the ddname. If multiple active data sets are to be read, use DDNAMEs ACTIVE1, ACTIVE2, ... ACTIVE*n* to specify the data sets. Specify the data sets in ascending log RBA order with ACTIVE1 being the lowest RBA and ACTIVE*n* being the highest. |
| **DD statements for data sharing users** | |

*Table 170. JCL DD statements for DB2 stand-alone log services (continued)*

| JCL DD statement | Explanation |
|---|---|
| GROUP | If you are reading logs from every member of a data sharing group in LRSN sequence, you can use this statement to locate the BSDSs and log data sets needed. You must include the data set name of one BSDS in the statement. DB2 can find the rest of the information from that one BSDS.<br>All members' logs and BSDS data sets must be available. If you use this DD statement, you must also use the LRSN and RANGE parameters on the OPEN request. The GROUP DD statement overrides any M*xx*BSDS statements that are used.<br>(DB2 searches for the BSDS DD statement first, then the GROUP statement, and then the M*xx*BSDS statements. If for some reason you want to use a particular member's BSDS for your own processing, you must call that DD statement something other than BSDS.) |
| M*xx*BSDS | Names the BSDS data set of a member whose log must participate in the read operation and whose BSDS is to be used to locate its log data sets. Use a separate M*xx*BSDS DD statement for each DB2 member. *xx* can be any 2 valid characters.<br>Use these statements if logs from selected members of the data sharing group are required and the BSDSs of those members are available. These statements are ignored if you use the GROUP DD statement.<br>For one M*xx*BSDS statement, you can use either RBA or LRSN values to specify a range. If you use more than one M*xx*BSDS statement, you must use the LRSN to specify the range. |
| M*yy*ARCHV | Names the archive log data sets of a member to be used as input. *yy* can be any 2 valid characters that do not duplicate any *xx* used in an M*xx*BSDS DD statement.<br>Concatenate all required archived log data sets of a given member in time sequence under one DD statement. Use a separate M*yy*ARCHV DD statement for each member. You must use this statement if the BSDS data set is unavailable or if you want only some of the log data sets from selected members of the group.<br>If you name the BSDS of a member by a M*xx*BSDS DD statement, do not name the log of the same member by an M*yy*ARCHV statement. If both M*yy*ARCHV and M*xx*BSDS identify the same log data sets, the service request fails. M*yy*ARCHV statements are ignored if you use the GROUP DD statement. |
| M*yy*ACT*n* | Names the active log data set of a member to be used as input. *yy* can be any 2 valid characters that do not duplicate any *xx* used in an M*xx*BSDS DD statement. Use the same characters that identify the M*yy*ARCHV statement for the same member; do *not* use characters that identify the M*yy*ARCHV statement for any other member. *n* is a number from 1 to 16. Assign values of *n* in the same way as for ACTIVE*n* DD statements.<br>You can use this statement if the BSDS data sets are unavailable or if you want only some of the log data sets from selected members of the group.<br>If you name the BSDS of a member by a M*xx*BSDS DD statement, do not name the log of the same member by an M*yy*ACT*n* statement. M*yy*ACT*n* statements are ignored if you use the GROUP DD statement. |

The DD statements must specify the log data sets in ascending order of log RBA (or LRSN) range. If both ARCHIVE and ACTIVE*n* DD statements are included, the first archive data set must contain the lowest log RBA or LRSN value. If the JCL specifies the data sets in a different order, the job terminates with an error return code with a GET request that tries to access the first record breaking the sequence. If the log ranges of the two data sets overlap, this is not considered an error; instead, the GET function skips over the duplicate data in the second data set and returns the next record. The distinction between out-of-order and overlap is as follows:

- **Out-of-order condition** occurs when the log RBA or LRSN of the first record in a data set is greater than that of the first record in the following data set.
- **Overlap condition** occurs when the out-of-order condition is not met but the log RBA or LRSN of the last record in a data set is greater than that of the first record in the following data set.

Gaps within the log range are permitted. A gap is created when one or more log data sets containing part of the range to be processed are not available. This can happen if the data set was not specified in the JCL or is not reflected in the BSDS. When the gap is encountered, an exception return code value is set, and the next complete record following the gap is returned.

Normally, the BSDS ddname will be supplied in the JCL, rather than a series of ACTIVE ddnames or a concatenated set of data sets for the ARCHIVE ddname. This is commonly referred to as “running in BSDS mode”.

## Data sharing users: Which members participate in the read?

The number of members whose logs participate in a particular read request is determined by:

- The number of members in the group, if you use the GROUP DD statement
- Otherwise, the number of different *xx*s and *yy*s used in the M*xx* and M*yy* type DD statements.

For example, assume you need to read log records from members S1, S2, S3, S4, S5 and S6.

S1 and S2 locate their log data sets by their BSDSs.
S3 and S4 need both archive and active logs.
S4 has two active log data sets.
S5 needs only its archive log.
S6 needs only one of its active logs.

Then you need the following DD statements to specify the required log data sets:

| | | | | | |
|---|---|---|---|---|---|
| MS1BSDS | MS2BSDS | MS3ARCHV | MS4ARCHV | MS5ARCHV | MS6ACT1 |
| | | MS3ACT1 | MS4ACT1 | | |
| | | | MS4ACT2 | | |

The order of the DD statements in the JCL stream is not important.

## Registers and return codes

The request macro invoking these services can be used by reentrant programs. The macro requires that register 13 point to an 18-word save area at invocation. In addition, registers 0, 1, 14, and 15 are used as work and linkage registers. A return code is passed back in register 15 at the completion of each request. When the return code is nonzero, a reason code is placed in register 0. Return codes identify a class of errors, while the reason code identifies a specific error condition of that class. The stand-alone log return codes are shown in Table 171.

*Table 171. Stand-alone log return codes*

| Return code | Explanation |
|---|---|
| 0 | Successful completion. |
| 4 | Exception condition (for example, end of file), not an error. This return code is not applicable for OPEN and CLOSE requests. |
| 8 | Unsuccessful completion due to improper user protocol. |
| 12 | Unsuccessful completion. Error encountered during processing of a valid request. |

The stand-alone log services invoke executable macros that can execute only in 24-bit addressing mode and reference data below the 16MB line. User-written applications should be link-edited as AMODE(24), RMODE(24).

## Stand-alone log OPEN request

Issue this request when you want to initialize the stand-alone log services. The syntax for the stand-alone log OPEN request is:

```
{label} DSNJSLR   FUNC=OPEN
                  ,LRSN=YES³NO
                  ,DDNAME= address or (Reg. 2-12) optional
                  ,RANGE= address or (Reg. 2-12) optional
                  ,PMO=CI or RECORD
```

**Keyword**
**Explanation**

**FUNC=OPEN**
Requests the stand-alone log OPEN function.

**LRSN** Tells DB2 how to interpret the log range:
NO: the log range is specified as RBA values. This is the default.
YES: the log range is specified as LRSN values.

**DDNAME**
Specifies the address of an 8-byte area which contains the ddname to be used as an alternate to a ddname of the BSDS when the BSDS is opened, or a register that contains that address.

**RANGE**
Specifies the address of a 12-byte area containing the log range to be processed by subsequent GET requests against the request block generated by this request, or a register that contains that address.

If LRSN=NO, then the range is specified as RBA values. If LRSN=YES, then the range is specified as LRSN values.

The first 6 bytes contain the low RBA or LRSN value. The first complete log record with an RBA or LRSN value equal to or greater than this value is the record accessed by the first log GET request against the request block. The last 6 bytes contain the end of the range or high RBA or LRSN value. An end-of-data condition is returned when a GET request tries to access a record with a starting RBA or LRSN value greater than this value. A value of 6 bytes of X'FF' indicates that the log is to be read until either the end of the log (as specified by the BSDS) or the end of the data in the last JCL-specified log data set is encountered.

If BSDS, GROUP, or MxxBSDS DD statements are used for locating the log data sets to be read, the RANGE parameter is required. If the JCL determines the log data sets to be read, the RANGE parameter is optional.

**PMO** Specifies the processing mode. You can use OPEN to retrieve either log records or control intervals in the same manner. Specify PMO=CI or RECORD, then use GET to return the data you have selected. The default is RECORD.

The rules remain the same regarding control intervals and the range specified for the OPEN function. Control intervals must fall within the range specified on the RANGE parameter.

**Output Explanation**

**GPR 1**
General-purpose register 1 contains the address of a request block on return from this request. This address must be used for subsequent stand-alone log requests. When no more log GET operations are required by the program, this request block should be used by a FUNC=CLOSE request.

**GPR 15**
General-purpose register 15 contains a return code upon completion of a request. For nonzero return codes, a corresponding reason code is contained in register 0. The return codes are listed and explained in Table 171 on page 975.

**GPR 0**
General-purpose register 0 contains a reason code associated with a nonzero return code in register 15.

See Part 3 of *DB2 Messages and Codes* for reason codes that are issued with the return codes.

***Log control interval retrieval:*** You can use the PMO option to retrieve log control intervals from archive log data sets. DSNJSLR also retrieves log control intervals from the active log if the DB2 system is not active. During OPEN, if DSNJSLR detects that the control interval range is not within the archive log range available (for example, the range purged from BSDS), an error condition is returned.

Specify CI and use GET to retrieve the control interval you have chosen. The rules remain the same regarding control intervals and the range specified for the OPEN function. Control intervals must fall within the range specified on the RANGE parameter.

***Log control interval format:*** A field in the last 7 bytes of the control interval, offset 4090, contains a 7-byte timestamp. This field reflects the time at which the control interval was written to the active log data set. The timestamp is in store clock (STCK) format and is the high order 7 bytes of the 8-byte store clock value.

## Stand-alone log GET request

This request returns a pointer to a buffer containing the next log record based on position information in the request block.

A log record is available in the area pointed to by the request block until the next GET request is issued. At that time, the record is no longer available to the requesting program. If the program requires reference to a log record's content after requesting a GET of the next record, the program must move the record into a storage area that is allocated by the program.

The first GET request, after a FUNC=OPEN request that specified a RANGE parameter, returns a pointer in the request feedback area. This points to the first record with a log RBA value greater than or equal to the low log RBA value specified by the RANGE parameter. If the RANGE parameter was not specified on the FUNC=OPEN request, then the data to be read is determined by the JCL specification of the data sets. In this case, a pointer to the first complete log record in the data set that is specified by the ARCHIVE, or by ACTIVE1 if ARCHIVE is omitted, is returned. The next GET request returns a pointer to the next record in ascending log RBA order. Subsequent GET requests continue to move forward in log RBA sequence until the function encounters the end of RANGE RBA value, the end of the last data set specified by the JCL, or the end of the log as determined by the bootstrap data set.

The syntax for the stand-alone log GET request is:

```
{label} DSNJSLR   FUNC=GET
                  ,RBR=(Reg. 1-12)
```

**Keyword**
**Explanation**

**FUNC=GET**
Requests the stand-alone log GET function.

**RBR** Specifies a register that contains the address of the request block this request is to use. Although you can specify any register between 1 and 12, using register 1 (RBR=(1)) avoids the generation of an unnecessary load register and is therefore more efficient. The pointer to the request block (that is passed in register *n* of the RBR=(n) keyword) must be used by subsequent GET and CLOSE function requests.

**Output Explanation**

**GPR 15**
General-purpose register 15 contains a return code upon completion of a request. For nonzero return codes, a corresponding reason code is contained in register 0. Return codes are listed and explained in Table 171 on page 975.

**GPR 0**
General-purpose register 0 contains a reason code associated with a nonzero return code in register 15.

See Part 3 of *DB2 Messages and Codes* for reason codes that are issued with the return codes.

Reason codes 00D10261 - 00D10268 reflect a damaged log. In each case, the RBA of the record or segment in error is returned in the stand-alone feedback block field (SLRFRBA). A damaged log can impair DB2 restart; special recovery procedures are required for these circumstances. For recovery from these errors, refer to “Chapter 22. Recovery scenarios” on page 409.

Information about the GET request and its results is returned in the request feedback area, starting at offset X'00'. If there is an error in the length of some record, the control interval length is returned at offset X'0C' and the address of the beginning of the control interval is returned at offset X'08'.

On return from this request, the first part of the request block contains the feedback information that this function returns. Mapping macro DSNDSLRF defines the feedback fields which are shown in Table 172. The information returned is status information, a pointer to the log record, the length of the log record, and the 6-byte log RBA value of the record.

*Table 172. Stand-alone log get feedback area contents*

| Field name | Hex offset | Length (bytes) | Field contents |
|---|---|---|---|
| SLRFRC | 00 | 2 | Log request return code |
| SLRFINFO | 02 | 2 | Information code returned by dynamic allocation. Refer to the MVS SPF job management publication for information code descriptions |
| SLRFERCD | 04 | 2 | VSAM or dynamic allocation error code, if register 15 contains a nonzero value. |
| SLRFRG15 | 06 | 2 | VSAM register 15 return code value. |
| SLRFFRAD | 08 | 4 | Address of area containing the log record or CI |
| SLRFRCLL | 0C | 2 | Length of the log record or RBA |
| SLRFRBA | 0E | 6 | Log RBA of the log record |
| SLRFDDNM | 14 | 8 | ddname of data set on which activity occurred |

## Stand-alone log CLOSE request

This request deallocates any log data sets that were dynamically allocated by previous processing. Also, all storage that was obtained by previous functions, including the request block specified on this request, is freed.

The syntax for the stand-alone log CLOSE request is:

```
{label} DSNJSLR   FUNC=CLOSE
                  ,RBR=(Reg. 1-12)
```

**Keyword**
**Explanation**

**FUNC=CLOSE**
Requests the CLOSE function.

**RBR** Specifies a register that contains the address of the request block that this function uses. Although you can specify any register between 1 and 12, using register 1 (RBR=(1)) avoids the generation of an unnecessary load register and is therefore more efficient.

**Output Explanation**

**GPR 15**
Register 15 contains a return code upon completion of a request. For nonzero return codes, a corresponding reason code is contained in register 0. The return codes are listed and explained in Table 171 on page 975.

**GPR 0**
Register 0 contains a reason code that is associated with a nonzero return code that is contained in register 15. The only reason code used by the CLOSE function is 00D10030.

See Part 3 of *DB2 Messages and Codes* for reason code details.

## Sample application program using stand-alone log services

Figure 140 shows sample segments of an assembler program that uses the three stand-alone log services (OPEN, GET, and CLOSE) to process one log record.

```
TSTJSLR5 CSECT
   ⋮

OPENCALL EQU   *
         LA    R2,NAME                GET BSDS DDNAME ADDRESS
         LA    R3,RANGER              GET ADDRESS OF RBA RANGE
         DSNJSLR FUNC=OPEN,DDNAME=(R2),RANGE=(R3)
         LTR   R15,R15                CHECK RETURN CODE FROM OPEN
         BZ    GETCALL                OPEN OK, DO GET CALLS
   ⋮
```

*Figure 140. Excerpts from a sample program using stand-alone log services (Part 1 of 4)*

```
*****************************************************************
*        HANDLE ERROR FROM OPEN FUNCTION AT THIS POINT          *
*****************************************************************
   ⋮

GETCALL  EQU   *
         DSNJSLR FUNC=GET,RBR=(R1)
         C     R0,=X'00D10020'        END OF RBA RANGE ?
         BE    CLOSE                  YES, DO CLEANUP
         C     R0,=X'00D10021'        RBA GAP DETECTED ?
         BE    GAPRTN                 HANDLE RBA GAP
         LTR   R15,R15                TEST RETURN CODE FROM GET
         BNZ   ERROR
   ⋮
```

*Figure 140. Excerpts from a sample program using stand-alone log services (Part 2 of 4)*

```
   ⋮

******************************************************************
*       PROCESS RETURNED LOG RECORD AT THIS POINT. IF LOG RECORD *
*       DATA MUST BE KEPT ACROSS CALLS, IT MUST BE MOVED TO A    *
*       USER-PROVIDED AREA.                                      *
******************************************************************
         USING SLRF,1                 BASE SLRF DSECT
         L     R8,SLRFFRAD            GET LOG RECORD START ADDR
         LR    R9,R8
         AH    R9,SLRFRCLL            GET LOG RECORD END ADDRESS
         BCTR  R9,R0
   ⋮
```

*Figure 140. Excerpts from a sample program using stand-alone log services (Part 3 of 4)*

```
CLOSE    EQU   *
         DSNJSLR FUNC=CLOSE,RBR=(1)
   ⋮

NAME     DC    C'DDBSDS'
RANGER   DC    X'00000000000000000005FFFF'
   ⋮

         DSNDSLRB
         DSNDSLRF
         EJECT
R0       EQU   0
R1       EQU   1
R2       EQU   2
   ⋮

R15      EQU   15
         END
```

*Figure 140. Excerpts from a sample program using stand-alone log services (Part 4 of 4)*

## Reading log records with the log capture exit

You can use the log capture exit to capture DB2 log data in real time. This installation exit presents log data to a log capture exit routine when the data is written to the DB2 active log. This exit is not intended to be used for general purpose log auditing or tracking. The IFI interface (see “Reading log records with IFI” on page 968) is designed (and is more appropriate) for this purpose.

The log capture exit executes in an area of DB2 that is critical for performance. As such, it is primarily intended as a mechanism to capture log data for recovery purposes such as with the Remote Recovery Data Facility (RRDF) Release 2 program offering. In addition, the log capture exit operates in a very restrictive MVS environment, which severely limits its capabilities as a stand-alone routine.

To capture log records with this exit, you must first write an exit routine (or use the one provided by the program offering mentioned above) that can be loaded and called under the various processing conditions and restrictions required of this exit. See “Log capture routines” on page 944 and refer to the previous sections of this appendix, “What the log contains” on page 957 and “The physical structure of the log” on page 962.

# Appendix D. Interpreting DB2 trace output

The information in this appendix is Product-sensitive Programming Interface and Associated Guidance Information as defined in “Notices” on page 1095.

When you activate a DB2 trace, it produces trace records based on the parameters you specified for the -START TRACE command. Each record identifies one or more significant DB2 events. You can use DB2 Performance Monitor (DB2 PM), a separately licensed program, to format, print, and interpret DB2 trace output. However, if you do not have DB2 PM or you want to do your own analysis of the trace output, you can use the information in this appendix and the trace field descriptions which are shipped to you as part of the DB2 product. By examining a DB2 trace record, you can determine the type of trace that produced the record (statistics, accounting, audit, performance, monitor, or global) and the event the record reports.

Please note that where the trace output indicates a particular release level, you will see 'xx' to show that this information varies according to the actual release of DB2 that you are using.

## Processing trace records

Trace records can be written to SMF or GTF. Regardless of whether you write the record to SMF or GTF, it contains up to four basic sections:
- An SMF or GTF writer header section
- A self-defining section
- A product section
- Zero or more data sections

Figure 141 shows the format of DB2 trace records.

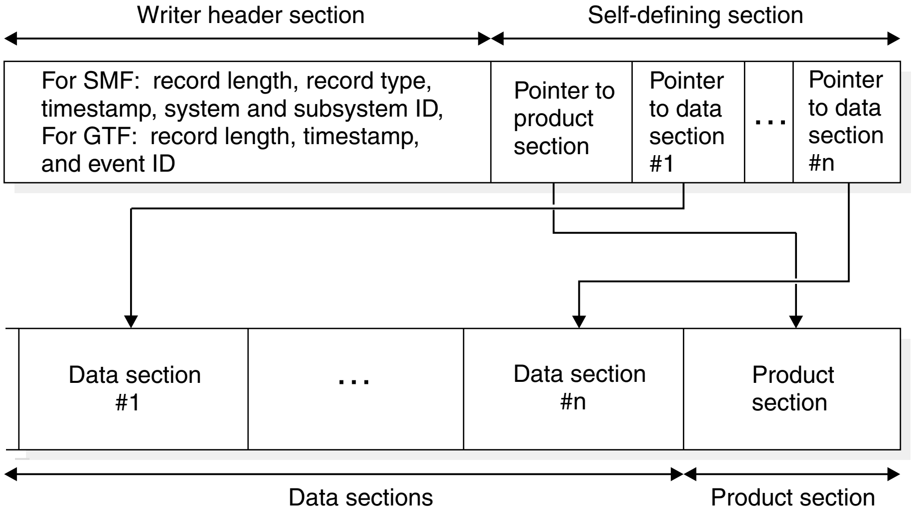

Writer header section
Self-defining section
For SMF: record length, record type, timestamp, system and subsystem ID, For GTF: record length, timestamp, and event ID
Pointer to product section
Pointer to data section #1
...
Pointer to data section #n
Data section #1
...
Data section #n
Product section
Data sections
Product section

*Figure 141. General format of trace records written by DB2*

The writer header section begins at the first byte of the record and continues for a fixed length. (The GTF writer header is longer than the SMF writer header.)

The self-defining section follows the writer header section (both GTF and SMF) and is further described in “Self-defining section” on page 988. The first self-defining section always points to a special data section called the product section. Among other things, the product section contains an instrumentation facility component identifier (IFCID). Descriptions of the records differ for each IFCID. For a list of records, by IFCID, for each class of a trace, see the description of the START TRACE command in *DB2 Command Reference*.

To interpret a record, find its description, by IFCID, in one of the following mapping macros:

| IFCID | Mapped by Macro |
|---|---|
| 0001 | DSNDQWST, subtype=0 |
| 0002 | DSNDQWST, subtype=1 |
| 0003 | DSNDQWAS |
| 0004—0057 | DSNDQW00 |
| 0058—0139 (except 0106) | DSNDQW01 |
| 0106 | DSNDQWPZ |
| 0140—196, 198, 199 | DSNDQW02 |
| 0201—0249 (except 0202, 230 and 239) | DSNDQW03 |
| 0202 | DSNDQWS2, subtype=2 |
| 0230 | DSNDQWST, subtype=3 |
| 0239 | DSNDQWAS and DSNDQWA1 |
| 0250—0330 | DSNDQW04 |

The product section also contains field QWHSNSDA, which indicates how many self-defining data sections the record contains. You can use this field to keep from trying to access data sections that do not exist. In trying to interpret the trace records, remember that the various keywords you specified when you started the trace determine whether any data is collected. If no data has been collected, field QWHSNSDA shows a data length of zero.

## SMF writer header section

In SMF, writer headers for statistics records are mapped by macro DSNDQWST, for accounting records by DSNDQWAS, and for performance, audit, and monitor records by DSNDQWSP. When these macros are assembled, they include the other macros necessary to map the remainder of the trace records sent to SMF.

The SMF writer header section begins at the first byte of the record. After establishing addressability, you can examine the header fields. The fields are described in Table 173. Figure 142 on page 983 is a sample of the first record of DB2 performance trace output sent to SMF.

*Table 173. Contents of SMF writer header section*

| Hex Offset | Macro DSNDQWST, statistics field | Macro DSNDQWAS, accounting field | Macro DSNDQWSP, monitor, audit, and performance field | Description |
|---|---|---|---|---|
| 0 | SM100LEN | SM101LEN | SM102LEN | Total length of SMF record |
| 2 | SM100SGD | SM101SGD | SM102SGD | Segment descriptor |

*Table 173. Contents of SMF writer header section (continued)*

| Hex Offset | Macro DSNDQWST, statistics field | Macro DSNDQWAS, accounting field | Macro DSNDQWSP, monitor, audit, and performance field | Description |
|---|---|---|---|---|
| 4 | SM100FLG | SM101FLG | SM102FLG | System indicator |
| 5 | SM100RTY | SM101RTY | SM102RTY | SMF record type<br>Statistics=100(dec), Accounting=101(dec), Monitor=102(dec), Audit=102(dec), Performance=102(dec) |
| 6 | SM100TME | SM101TME | SM102TME | SMF record timestamp, time portion |
| A | SM100DTE | SM101DTE | SM102DTE | SMF record timestamp, date portion |
| E | SM100SID | SM101SID | SM102SID | System ID |
| 12 | SM100SSI | SM101SSI | SM102SSI | Subsystem ID |
| 16 | SM100STF | SM101STF | SM102STF | Reserved |
| 17 | SM100RI | SM101RI | SM102RI | Reserved |
| 18 | SM100BUF | SM101BUF | SM102BUF | Reserved |
| 1C | SM100END | SM101END | SM102END | End of SMF header |

```
          A         B   C       D        E          F                     G  H
000000   01240000 0E660030  9EEC0093 018FF3F0   F9F0E2E2 D6D70000  00000000 0000008C
         I    J    K        L   M    N
000020   00980001 0000002C  005D0001 00550053   4DE2E3C1 D9E340E3  D9C1C3C5 404DE2E3
000040   C1E3405D C3D3C1E2  E2404D5C 405DD9D4   C9C4404D 5C405DD7  D3C1D540 4D5C405D
000060   C1E4E3C8 C9C4404D  5C405DC9 C6C3C9C4   404D5C40 5DC2E4C6  E2C9E9C5 404D5C40
                                     O          P   Q  R
000080   5D000000 01000101  01000000 004C0110   000402xx 00B3AB78  E2E2D6D7 A6E9BACB
                                                S
0000A0   F6485E02 00000003  00000021 00000001   E2C1D5E3 C16DE3C5  D9C5E2C1 6DD3C1C2
0000C0   C4C2F2D5 C5E34040  D3E4D5C4 F0404040   A6E9BACB F4570001  004C0200 E2E8E2D6
0000E0   D7D94040 F0F2F34B  C7C3E2C3 D5F6F0F2   E2E2D6D7 40404040  40404040 40404040
000100   E2E8E2D6 D7D94040  00000000 00000000   00000000 00000000  00000000 00000000
000120   00000000 T
```

*Figure 142. DB2 trace output sent to SMF (printed with DFSERA10 print program of IMS)*

| Key to Figure 142 | Description |
|---|---|
| A 0124 | Record length (field SM102LEN); beginning of SMF writer header section |
| B 66 | Record type (field SM102RTY) |
| C 0030 9EEC | Time (field SM102TME) |
| D 0093 018F | Date (field SM102DTE) |
| E F3F0 F9F0 | System ID (field SM102SID) |
| F E2E2 D6D7 | Subsystem ID (field SM102SSI) |
| G | End of SMF writer header section |
| H 0000008C | Offset to product section; beginning of self-defining section |
| I 0098 | Length of product section |
| J 0001 | Number of times the product section is repeated |
| K 0000002C | Offset to first (in this case, only) data section |
| L 005D | Length of data section |

| Key to Figure 142 on page 983 | Description |
|---|---|
| M 0001 | Number of times the data section is repeated |
| N 00550053 | Beginning of data section |
| O | Beginning of product section |
| P 0004 | IFCID (field QWHSIID) |
| Q 02 | Number of self-defining sections in the record (field QWHSNSDA) |
| R xx | Release indicator number (field QWHSRN); this varies according to the actual level of DB2 you are using. |
| S E2C1D5E3... | Local location name (16 bytes) |
| T | End of first record |

## GTF writer header section

The length and content of the writer header section differs between SMF and GTF records, but the other sections of the records are the same for SMF and GTF.

The GTF writer header section begins at the first byte of the record. After establishing addressability, you can examine the fields of the header. The writer headers for trace records sent to GTF are always mapped by macro DSNDQWGT. The fields are described in Table 174.

*Table 174. Contents of GTF writer header section*

| Offset | Macro DSNDQWGT field | Description |
|---|---|---|
| 0 | QWGTLEN | Length of Record |
| 2 | | Reserved |
| 4 | QWGTAID | Application identifier |
| 5 | QWGTFID | Format ID |
| 6 | QWGTTIME | Timestamp; you must specify TIME=YES when you start GTF. |
| 14 | QWGTEID | Event ID: X'EFB9' |
| 16 | QWGTASCB | ASCB address |
| 20 | QWGTJOBN | Job name |
| 28 | QWGTHDRE | Extension to header |
| 28 | QWGTDLEN | Length of data section |
| 30 | QWGTDSCC | Segment control code<br>0=Complete 2=Last 1=First 3=Middle |
| 31 | QWGTDZZ2 | Reserved |
| 32 | QWGTSSID | Subsystem ID |
| 36 | QWGTWSEQ | Sequence number |
| 40 | QWGTEND | End of GTF header |

Figure 143 on page 985 contains trace output sent to GTF.

```
DFSERA10 - PRINT PROGRAM

000000   001A0000 0001FFFF  94B6A6E9 BD6636FA   5C021000 00010000  0000
         A            B                  C               D                  E   F
000000   011C0000 FF00A6E9  C33E28F7 DD03EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000100
         G               H   I        J    K      L         M    N      O
000020   E2E2D6D7 00000001  000000A0 00980001   00000038 00680001  0060005E 4DE2E3C1
000040   D9E340E3 D9C1C3C5  404DE2E3 C1E3405D   C3D3C1E2 E2404D5C  405DD9D4 C9C4404D
000060   5C405DC4 C5E2E340  4DC7E3C6 405DD7D3   C1D5404D 5C405DC1  E4E3C8C9 C4404D5C
000080   405DC9C6 C3C9C440  4D5C405D C2E4C6E2   C9E9C540 4D5C405D  FFFFFFFF 00040101
          P        Q   R  S
0000A0   004C0110 000402xx  00B3ADB8 E2E2D6D7   A6E9C33E 28EF4403  00000006 00000001
                  T
0000C0   00000001 E2C1D5E3  C16DE3C5 D9C5E2C1   6DD3C1C2 C4C2F2D5  C5E34040 D3E4D5C4
0000E0   F0404040 A6E9C33E  271F0001 004C0200   E2E8E2D6 D7D94040  F0F2F34B C7C3E2C3
000100   D5F6F0F2 E2E2D6D7  40404040 40404040   40404040 E2E8E2D6  D7D94040
                                                                                U
000000   00440000 FF00A6E9  C33E2901 1303EFB9   00F91400 E2E2D6D7  D4E2E3D9 00280200
000020   E2E2D6D7 00000001  00000000 00000000   00000000 00000000  00000000 00000000
000040   00000000 V
        W                                                                       X
000000   011C0000 FF00A6E9  C33E2948 E203EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000100
000020   E2E2D6D7 00000002  000006D8 004C0001   00000090 001C0004  00000100 001C000E
000040   00000288 0018000E  00000590 00400001   000005D0 00740001  00000480 00440001
000060   000003D8 00800001  00000458 00280001   00000644 00480001  000004E4 00AC0001
000080   0000068C 004C0001  000004C4 00200001   D4E2E3D9 00000001  762236F2 00000000
0000A0   59F48900 001E001E  00F91400 C4C2D4F1   00000001 1A789573  00000000 95826100
0000C0   001F001F 00F90E00  C4C9E2E3 00000000   3413C60E 00000000  1C4D0A00 00220022
0000E0   00F90480 C9D9D3D4  00000000 0629E2BC   00000000 145CE000  001D001D 00F91600
000100   E2D4C640 00000046  00000046 00000000   00000000 00000000  00000000
                                                                                 Y
000000   011C0000 FF00A6E9  C33E294B 1603EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000300
000020   E2E2D6D7 00000002  D9C5E240 00000000   00000000 00000000  00000000 00000000
000040   00000000 C7E3C640  00000001 00000001   00000000 00000000  00000000 00000000
000060   E2D9E540 00000000  00000000 00000000   00000000 00000000  00000000 E2D9F140
000080   00000156 000000D2  00000036 00000036   00000000 00000004  E2D9F240 00000000
0000A0   00000000 00000000  00000000 00000000   00000000 D6D7F140  00000000 00000000
0000C0   00000000 00000000  00000000 00000000   D6D7F240 00000000  00000000 00000000
0000E0   00000000 00000000  00000000 D6D7F340   00000000 00000000  00000000 00000000
000100   00000000 00000000  D6D7F440 00000000   00000000 00000000  00000000
                                                                                 Y
000000   011C0000 FF00A6E9  C33E294D 3C03EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000300
000020   E2E2D6D7 00000002  00000000 00000000   D6D7F540 00000000  00000000 00000000
000040   00000000 00000000  00000000 D6D7F640   00000000 00000000  00000000 00000000
000060   00000000 00000000  D6D7F740 00000000   00000000 00000000  00000000 00000000
000080   00000000 D6D7F840  00000000 00000000   00000000 00000000  00000000 00000000
0000A0   00010000 0000000E  0000000D 00000000   00000000 00000000  00020000 0000000D
0000C0   0000000D 00000000  00000000 00000000   00030000 00000003  00000003 00000000
0000E0   00000000 00000000  00040000 00000006   00000006 00000000  00000000 00000000
000100   00050000 00000005  00000005 00000000   00000000 00000000  006A0000
                                                                                 Y
000000   011C0000 FF00A6E9  C33E294F 6103EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000300
000020   E2E2D6D7 00000002  00000005 00000005   00000000 00000000  00000000 008C0000
000040   00000000 00000000  00000000 00000000   00000000 008D0000  00000000 00000000

  ⋮

                                                                                 Z
000000   00780000 FF00A6E9  C33E2957 D103EFB9   00F91400 E2E2D6D7  D4E2E3D9 005C0200
                                      AA
000020   E2E2D6D7 00000002  00000000 004C011A   00010D31 02523038  E2E2D6D7 A6E9C33E
000040   29469A03 0000000E  00000002 00000001   E2C1D5E3 C16DE3C5  D9C5E2C1 6DD3C1C2
000060   40404040 40404040  40404040 40404040   A6E9B6B4 9A2B0001
```

*Figure 143. DB2 trace output sent to GTF (spanned records printed with DFSERA10 print program of IMS)*

| Key to Figure 143 on page 985 | Description |
|---|---|
| A 011C | Record length (field QWGTLEN); beginning of GTF writer header section |
| B A6E9 C33E28F7 DD03 | Timestamp (field QWGTTIME) |
| C EFB9 | Event ID (field QWGTEID) |
| D E2E2D6D7 D4E2E3D9 | Job name (field QWGTJOBN) |
| E 0100 | Length of data section |
| F 01 | Segment control code (01 = first segment of the first record) |
| G E2E2D6D7 | Subsystem ID (field QWGTSSID) |
| H | End of GTF writer header section |
| I 000000A0 | Offset to product section; beginning of self-defining section |
| J 0098 | Length of product section |
| K 0001 | Number of times the product section is repeated |
| L 00000038 | Offset to first (in this case, only) data section |
| M 0068 | Length of data section |
| N 0001 | Number of times the data section is repeated |
| O 0060005E | Beginning of data section |
| P 004C0110... | Beginning of product section |
| Q 0004 | IFCID (field QWHSIID) |
| R 02 | Number of self-defining sections in the record (field QWHSNSDA) |
| S xx | Release indicator number (field QWHSRN); this varies according to the actual release level of DB2 you are using. |
| T E2C1D5E3... | Local location name (16 bytes) |
| U 02 | Last segment of the first record |
| V | End of first record |
| W | Beginning of GTF header for new record |
| X 01 | First segment of a spanned record (QWGTDSCC = QWGTDS01) |
| Y 03 | Middle segment of a spanned record (QWGTDSCC = QWGTDS03) |
| Z 02 | Last segment of a spanned record (QWGTDSCC = QWGTDS02) |
| AA 004C | Beginning of product section |

GTF records are blocked to 256 bytes. Because some of the trace records exceed the GTF limit of 256 bytes, they have been blocked by DB2. Use the following logic to process GTF records:

1. Is the GTF event ID of the record equal to the DB2 ID (that is, does QWGTEID = X'*x*FB9')?

   If it is *not* equal, get another record.

   If it is equal, continue processing.
2. Is the record spanned?

   If it is spanned (that is, QWGTDSCC ¬ = QWGTDS00), test to determine whether it is the first, middle, or last segment of the spanned record.

   a. If it is the *first* segment (that is, QWGTDSCC = QWGTDS01), save the entire record including the sequence number (QWGTWSEQ) and the subsystem ID (QWGTSSID).

   b. If it is a *middle* segment (that is, QWGTDSCC = QWGTDS03), find the first segment matching the sequence number (QWGTSEQ) and on the subsystem ID (QWTGSSID). Then move the data portion immediately after the GTF header to the end of the previous segment.

c. If it is the *last* segment (that is, QWGTDSCC = QWGTDS02), find the first segment matching the sequence number (QWGTSEQ) and on the subsystem ID (QWTGSSID). Then move the data portion immediately after the GTF header to the end of the previous record.

   Now process the completed record.

If it is *not* spanned, process the record.

Figure 144 shows the same output after it has been processed by a user-written routine, which follows the logic outlined above.

```
000000   01380000 FF00A6E9  DCA7E275 1204EFB9   00F91400 E2E2D6D7  D4E2E3D9 011C0000
000020   E2E2D6D7 00000019  000000A0 00980001   00000038 00680001  0060005E 4DE2E3C1
000040   D9E340E3 D9C1C3C5  404DE2E3 C1E3405D   C3D3C1E2 E2404D5C  405DD9D4 C9C4404D
000060   5C405DC4 C5E2E340  4DC7E3C6 405DD7D3   C1D5404D 5C405DC1  E4E3C8C9 C4404D5C
000080   405DC9C6 C3C9C440  4D5C405D C2E4C6E2   C9E9C540 4D5C405D  00000001 00040101
0000A0   004C0110 000402xx  00B3ADB8 E2E2D6D7   0093018F 11223310  0000000C 00000019
0000C0   00000001 E2C1D5E3  C16DE3C5 D9C5E2C1   6DD3C1C2 C4C2F2D5  C5E34040 D3E4D5C4
0000E0   F0404040 A6E9DCA7  DF960001 004C0200   E2E8E2D6 D7D94040  F0F2F34B C7C3E2C3
000100   D5F6F0F2 E2E2D6D7  40404040 40404040   40404040 E2E8E2D6  D7D94040 00000000
000120   00000000 00000000  00000000 00000000   00000000 00000000
         A                                B
000000   07240000 FF00A6E9  DCA8060C 2803EFB9   00F91400 E2E2D6D7  D4E2E3D9 07080000
                         C  D                   E
000020   E2E2D6D7 0000001A  000006D8 004C0001   00000090 001C0004  00000100 001C000E
000040   00000288 0018000E  00000590 00400001   000005D0 00740001  00000480 00440001
000060   000003D8 00800001  00000458 00280001   00000644 00480001  000004E4 00AC0001
                            F
000080   0000068C 004C0001  000004C4 00200001   D4E2E3D9 00000003  27BCFDBC 00000000
0000A0   AB000300 001E001E  00F91400 C4C2D4F1   00000001 1DE8AEE2  00000000 DB0CB200
0000C0   001F001F 00F90E00  C4C9E2E3 00000000   4928674B 00000000  217F6000 00220022
0000E0   00F90480 C9D9D3D4  00000000 07165F79   00000000 3C2EF500  001D001D 00F91600
000100   E2D4C640 0000004D  0000004D 00000000   00000000 00000000  00000000 D9C5E240
000120   00000000 00000000  00000000 00000000   00000000 00000000  C7E3C640 00000019
000140   00000019 00000000  00000000 00000000   00000000 E2D9E540  00000000 00000000
000160   00000000 00000000  00000000 00000000   E2D9F140 00000156  000000D2 00000036
000180   00000036 00000000  00000004 E2D9F240   00000092 00000001  00000091 00000091
0001A0   00000000 0000000C  D6D7F140 00000002   00000001 00000001  00000000 00010000
0001C0   20000004 D6D7F240  00000000 00000000   00000000 00000000  00000000 00000000
0001E0   D6D7F340 00000000  00000000 00000000   00000000 00000000  00000000 D6D7F440
000200   00000000 00000000  00000000 00000000   00000000 00000000  D6D7F540 00000000
000220   00000000 00000000  00000000 00000000   00000000 D6D7F640  00000000 00000000
000240   00000000 00000000  00000000 00000000   D6D7F740 00000000  00000000 00000000
000260   00000000 00000000  00000000 D6D7F840   00000000 00000000  00000000 00000000
000280   00000000 00000000  00010000 00000042   00000011 00000030  00000000 00000000
0002A0   00020000 00000041  00000011 00000030   00000000 00000000  00030000 00000003
0002C0   00000003 00000000  00000000 00000000   00040000 0000000C  0000000C 00000000
0002E0   00000000 00000000  00050000 0000000B   0000000A 00000001  00000000 00000000
000300   006A0000 0000000C  0000000B 00000001   00000000 00000000  008C0000 00000000
000320   00000000 00000000  00000000 00000000   008D0000 00000000  00000000 00000000
000340   00000000 00000000  008E0000 00000000   00000000 00000000  00000000 00000000
```

*Figure 144. DB2 trace output sent to GTF (assembled with a user-written routine and printed with DFSERA10 print program of IMS) (Part 1 of 2)*

```
000360   008F0000 00000000  00000000 00000000   00000000 00000000  00900000 00000000
000380   00000000 00000000  00000000 00000000   00910000 00000000  00000000 00000000
0003A0   00000000 00000000  00920000 00000000   00000000 00000000  00000000 00000000
0003C0   00CA0000 00000041  00000011 00000030   00000000 00000000  00000000 00000000
0003E0   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000400   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000420   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000440   00000000 00000000  00000000 00000004   00000000 00000000  000005D4 00000130
000460   0000000D 0000000A  00000029 00000009   000000C3 00000000  00000000 00000000
000480   00000001 0000000C  00000000 04A29740   00000000 00000000  00000001 00000000
0004A0   00000001 00000000  00000000 00000000   00000000 00000000  00000000 00000000
0004C0   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
0004E0   00000000 E2C1D56D  D1D6E2C5 40404040   40404040 00000000  00000002 00000003
000500   00000000 000004A8  000005C7 00000000   00000001 00000003  00000003 00000000
000520   00000001 00000000  00000001 00000000   00000000 00000000  00000000 00000000
000540   00000002 00000001  00000000 00000000   00000000 00000000  00000000 00000000
000560   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000580   00000000 00000000  00000002 00000000   00000003 00000000  00000003 00000006
0005A0   00000000 00000000  00000000 00000000   00000005 00000003  00000000 00000000
0005C0   00000000 00000003  00000000 00000000   00000000 00000000  00000000 00000000
0005E0   00000000 00000000  0000000C 00000001   00000000 00000007  00000000 00000000
000600   00000000 00000000  00000000 00000001   00000000 00000000  00000000 00000000
000620   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000640   00000000 003C0048  D8E2E2E3 00000035   00000006 00000002  0000009E 0000002B
000660   00000078 00000042  00000048 000000EE   0000001B 0000007B  0000004B 00000000
000680   00000000 00000000  00000000 0093004C   D8D1E2E3 00000000  000000FC 0000000E
0006A0   00000000 00000000  0000009D 00000000   00000000 00000016  0000000F 00000018
                                                                   G
0006C0   00000000 00000000  00000000 00000000   00000000 00000000  004C011A 00010Dxx
0006E0   02523038 E2E2D6D7  0093018F 11223324   00000042 0000001A  00000001 E2C1D5E3
000700   C16DE3C5 D9C5E2C1  6DD3C1C2 40404040   40404040 40404040  40404040 A6E9B6B4
000720   9A2B0001 H
```

*Figure 144. DB2 trace output sent to GTF (assembled with a user-written routine and printed with DFSERA10 print program of IMS) (Part 2 of 2)*

| Key to Figure 144 on page 987 | Description |
|---|---|
| A 0724 | Length of assembled record; beginning of GTF writer header section of second record (field QWGTLEN) |
| B EFB9 | GTF event ID (field QWGTEID) |
| C | End of GTF writer header section of second record |
| D 000006D8 | Offset to product section |
| E 00000090 | Offset to first data section |
| F 000004C4 | Offset to last data section |
| G 004C011A | Beginning of product section |
| H | End of second record |

## Self-defining section

The self-defining section following the writer header contains pointers that enable you to find the product and data sections, which contain the actual trace data.

Each “pointer” is a descriptor containing 3 fields, which are:

1. A fullword containing the offset from the beginning of the record to the data section.
2. A halfword containing the length of each item in the data section.
3. A halfword containing the number of times the data section is repeated. If that field contains “0”, the data section is not in the record. If it contains a number greater than 1, multiple data items are stored contiguously within that data

section. To find the second data item, add the length of the first data item to the address of the first data item (and so forth). Multiple data items within a specific data section always have the same length and format.

Pointers occur in a fixed order, and their meanings are determined by the IFCID of the record. Different sets of pointers can occur, and each set is described by a separate DSECT. Therefore, to examine the pointers, you must first establish addressability using the DSECT that provides the appropriate description of the self-defining section. To do this:

1. Compute the address of the self-defining section.

   The self-defining section begins at label “SM100END” for statistics records, “SM101END” for accounting records, and “SM102END” for performance and audit records. It does not matter which mapping DSECT you use, because the length of the SMF writer header is always the same.

   For GTF, use QWGTEND.

2. Determine the IFCID of the record.

   Use the first field in the self-defining section; it contains the offset from the beginning of the record to the product section. The product section contains the IFCID.

   The product section is mapped by DSNDQWHS; the IFCID is mapped by QWHSIID.

   For statistics records having IFCID 0001, establish addressability using label “QWS0”; for statistics records having IFCID 0002, establish addressability using label “QWS1”. For accounting records, establish addressability using label “QWA0”. For performance and audit records, establish addressability using label “QWT0”.

After establishing addressability using the appropriate DSECT, use the pointers in the self-defining section to locate the record’s data sections.

To help make your applications independent of possible future releases of DB2, always use the length values contained in the self-defining section rather than symbolic lengths that you may find in the macro expansions.

The relationship between the contents of the self-defining section “pointers” and the items in a data section is shown in Figure 145 on page 990.

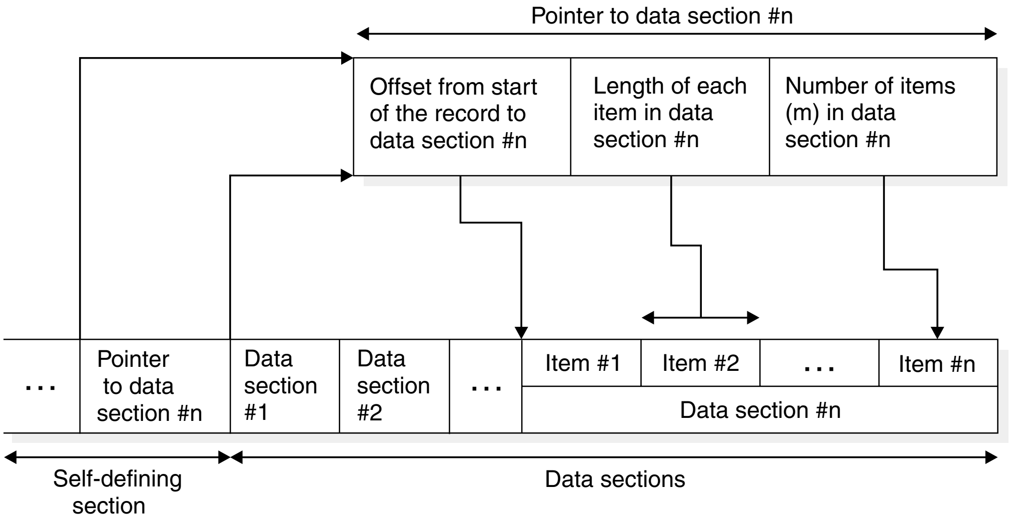

Pointer to data section #n
Offset from start of the record to data section #n
Length of each item in data section #n
Number of items (m) in data section #n
...
Pointer to data section #n
Data section #1
Data section #2
...
Item #1
Item #2
...
Item #n
Data section #n
Self-defining section
Data sections

*Figure 145. Relationship between self-defining section and data sections*

## Product section

The product section for all record types contains the standard header. The other headers—correlation, CPU, distributed, and data sharing data—may also be present.

*Table 175. Contents of product section standard header*

| Hex Offset | Macro DSNDQWHS field | Description |
|---|---|---|
| 0 | QWHSLEN | Length of standard header |
| 2 | QWHSTYP | Header type |
| 3 | QWHSRMID | RMID |
| 4 | QWHSIID | IFCID |
| 6 | QWHSRELN | Release number section |
| 6 | QWHSNSDA | Number of self-defining sections |
| 7 | QWHSRN | DB2 release identifier |
| 8 | QWHSACE | ACE address |
| C | QWHSSSID | Subsystem ID |
| 10 | QWHSSTCK | Timestamp—STORE CLOCK value assigned by DB2 |
| 18 | QWHSISEQ | IFCID sequence number |
| 1C | QWHSWSEQ | Destination sequence number |
| 20 | QWHSMTN | Active trace number mask |
| 24 | QWHSLOCN | Local location Name |
| 34 | QWHSLWID | Logical unit of work ID |
| 34 | QWHSNID | Network ID |
| 3C | QWHSLUNM | LU name |
| 44 | QWHSLUUV | Uniqueness value |

*Table 175. Contents of product section standard header (continued)*

| Hex Offset | Macro DSNDQWHS field | Description |
|---|---|---|
| 4A | QWHSLUCC | Commit count |
| 4C | QWHSEND | End of product section standard header |

*Table 176. Contents of product section correlation header*

| Hex Offset | Macro DSNDQWHC field | Description |
|---|---|---|
| 0 | QWHCLEN | Length of correlation header |
| 2 | QWHCTYP | Header type |
| 3 | | Reserved |
| 4 | QWHCAID | Authorization ID |
| C | QWHCCV | Correlation ID |
| 18 | QWHCCN | Connection name |
| 20 | QWHCPLAN | Plan name |
| 28 | QWHCOPID | Original operator ID |
| 30 | QWHCATYP | The type of system that is connecting |
| 34 | QWHCTOKN | Trace accounting token field |
| 4A | | Reserved |
| 4C | QWHCEUID | User ID of at the workstation for the end user |
| 5C | QWHCEUTX | Transaction name for the end user |
| 7C | QWHCEUWN | Workstation name for the end user |
| 8E | QWHCEND | End of product section correlation header |

*Table 177. Contents of CPU header*

| Hex Offset | Macro DSNDQWHU field | Description |
|---|---|---|
| 0 | QWHULEN | Length of CPU header |
| 2 | QWHUTYP | Header type |
| 3 | | Reserved |
| 4 | QWHUCPU | CPU time of MVS TCB or SRB dispatched |
| C | QWHUCNT | Count field reserved |
| E | QWHUEND | End of header |

*Table 178. Contents of distributed data header*

| Hex Offset | Macro DSNDQWHD field | Description |
|---|---|---|
| 0 | QWHDLEN | Length of the distributed header |
| 2 | QWHDTYP | Header type |

*Table 178. Contents of distributed data header (continued)*

| Hex Offset | Macro DSNDQWHD field | Description |
|---|---|---|
| 3 | | Reserved |
| 4 | QWHDRQNM | Requester location name |
| 14 | QWHDTSTP | Timestamp for DBAT trace record |
| 1C | QWHDSVNM | EXCSAT SRVNAM parameter |
| 2C | QWHDPRID | ACCRDB PRDID parameter |
| 30 | QWHDEND | End of distributed header |

*Table 179. Contents of trace header*

| Hex Offset | Macro DSNDQWHT field | Description |
|---|---|---|
| 0 | QWHTLEN | Length of the trace header |
| 2 | QWHTTYP | Header type |
| 3 | | Reserved |
| 4 | QWHTTID | Event ID |
| 6 | QWHTTAG | ID specified on DSNWTRC macro |
| 7 | QWHTFUNC | Resource manager function code. Default is 0. |
| 8 | QWHTEB | Execution block address |
| C | QWHTPASI | Prior address space ID - EPAR |
| E | QWHTR14A | Register 14 address space ID |
| 10 | QWHTR14 | Contents of register 14 |
| 14 | QWHTR15 | Contents of register 15 |
| 18 | QWHTR0 | Contents of register 0 |
| 1C | QWHTR1 | Contents of register 1 |
| 20 | QWHTEXU | Address of MVS execution unit |
| 24 | QWHTDIM | Number of data items |
| 26 | QWHTHASI | Home address space ID |
| 28 | QWHTDATA | Address of the data |
| 2C | QWHTFLAG | Flags in the trace list |
| 2E | QWHTDATL | Length of the data list |
| 30 | QWHTEND | End of header |

*Table 180. Contents of data sharing header*

| Hex Offset | Macro DSNDQWHA field | Description |
|---|---|---|
| 0 | QWHALEN | Length of the data sharing header |
| 2 | QWHATYP | Header type |
| 3 | | Reserved |

*Table 180. Contents of data sharing header (continued)*

| Hex Offset | Macro DSNDQWHA field | Description |
|---|---|---|
| 4 | QWHAMEMN | DB2 member name |
| C | QWHADSGN | DB2 data sharing group name |
| 14 | QWHAEND | End of header |

Figure 146 on page 994 is an actual sample of accounting trace for a distributed transaction sent to SMF.

```
                                                                             A
000000   065C0000 0E650030  C8AB0093 018FF3F0   F9F0E2E2 D6D70000  00000000 00000590
         B   C     D        E   F    G                   H                   I
000020   00CC0001 00000064  00E40001 0000046C   00E40001 00000550  00400001 00000414
                   J                  K                   L                  M
000040   00580001 00000148  00DC0001 00000224   01000001 00000000  00000000 00000324
                   N
000060   00F00001 A6E9BB19  BDF7AC04 A6E9BB31   D4221703 00000000  00ACCF00 00000000
000080   06582600 00000000  12C41000 00000000   19EA6A00 0000000C  40404040 40404040
0000A0   00000000 00000000  00000001 00000000   00000013 BC47FF09  00000000 051D0700
0000C0   00000000 0509B300  00000000 00000000   00000000 000BD200  00000008 00000000
0000E0   00000002 6ADF1503  00000000 00000000   00000000 00000000  00000000 00000000
000100   00000002 00000002  00000000 00000000   00000000 00000000  00000000 00000000
000120   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
                             O
000140   00000000 00030001  E2C1D56D D1D6E2C5   40404040 40404040  00000000 00000002
000160   00000003 00000000  000004A8 000005C7   00000000 00000001  00000003 00000003
000180   00000000 00000001  00000000 00000001   00000000 00000000  00000000 00000000
0001A0   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000002
0001C0   00000001 00000000  00000000 00000000   00000000 80000113  00000000 00000000
0001E0   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000200   00000000 00000000  00000000 00000000   00000000 00000002  00000000 C4E2D5F0
                   P
000220   F3F0F1F0 54C4E2D5  F0F3F0F1 F0E2C1D5   6DD1D6E2 C5404040  40404040 40C4C2F2
000240   D5C5E340 40D3E4D5  C4F14040 40E3E2D6   40404040 40C2C1E3  C3C84040 40E2E8E2
000260   C1C4D440 40404040  40E2E8E2 C1C4D440   40C4E2D5 C5E2D7D9  D9000000 00000000
000280   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
0002A0   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
0002C0   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
0002E0   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000300   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
                   Q
000320   00000000 0001003A  40404040 40404040   40404040 40404040  40404040 40404040
000340   40404040 40404040  4040C4E2 D5C5E2D4   F6F84040 40404040  40404040 14D7D8F5
000360   1525F5F4 00000008  A6E9BB2F 4A964600   A6E9BB30 7E95B704  00000013 BC41EF09
000380   00000000 058EA200  00000000 060DDB00   00000000 0516F700  00000006 00000000
0003A0   00000000 00000000  00000000 000BD200   00000002 6ADF1503  00000000 00000000
0003C0   00000000 00000000  00000000 00000000   00000002 00000002  00000000 00000000
0003E0   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
                                                          R
000400   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000420   00000000 00000000  00000003 00000000   00000000 00000000  00000000 00000000
000440   00000001 00000000  00000008 00000002   00000000 00000001  00000000 00000004
                                      S
000460   00000000 00000000  00000000 209500E4   D8E7E2E3 00000000  00000000 00000000
000480   00000000 00000001  00000001 00000001   00000001 00000000  00000000 00000000
0004A0   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
0004C0   00000000 00000000  00000000 00000000   00000003 00000000  00000000 00000000
0004E0   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000500   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000520   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
                                                 T
000540   00000000 00000000  00000000 00000000   00000000 0000000B  00000000 00000000
000560   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
                                                 U
000580   00000000 00000000  00000009 00000000   004C011A 00030961  00B3ADB8 E2E2D6D7
0005A0   A6E9BB31 E074C605  00000003 0000002D   00000002 E2C1D5E3  C16DE3C5 D9C5E2C1
```

*Figure 146. DB2 distributed data trace output sent to SMF (printed with DFSERA10 print program of IMS) (Part 1 of 2). In this example there is one accounting record (IFCID 0003) from the server site (SANTA_TERESA_LAB). The self-defining section for IFCID 0003 is mapped by DSNDQWA0.*

```
                                                                              V
0005C0   6DD3C1C2 C4C2F2D5  C5E34040 D3E4D5C4   F1404040 A6E9BAEC  C4D90002 008E0200
0005E0   E2E8E2C1 C4D44040  E2E8E2C1 C4D44040   40404040 E3E2D640  40404040 C4E2D5C5
000600   E2D7D9D9 E2E8E2C1  C4D44040 00000007   00000000 00000000  00000000 00000000
000620   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000640   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
                                  W
000660   00000000 E0000000  00000034 1000E2C1   D56DD1D6 E2C54040  40404040 4040A6E9
000680   BB2F38CC AC01E2C1  D56DD1D6 E2C54040   40404040 4040C4E2  D5F0F3F0 F1F0
```

*Figure 146. DB2 distributed data trace output sent to SMF (printed with DFSERA10 print program of IMS) (Part 2 of 2). In this example there is one accounting record (IFCID 0003) from the server site (SANTA_TERESA_LAB). The self-defining section for IFCID 0003 is mapped by DSNDQWA0.*

| Key to Figure 146 on page 994 | Description |
|---|---|
| A 00000590 | Offset to product section; beginning of self-defining section |
| B 00CC | Length of product section |
| C 0001 | Number of times product section is repeated |
| D 00000064 | Offset to accounting section |
| E 00E4 | Length of accounting section |
| F 0001 | Number of times accounting section is repeated |
| G 0000046C | Offset to SQL accounting section |
| H 00000550 | Offset to buffer manager accounting section |
| I 00000414 | Offset to locking accounting section |
| J 00000148 | Offset to distributed section |
| K 00000224 | Offset to MVS/DDF accounting section |
| L 00000000 | Offset to IFI accounting section |
| M 00000324 | Offset to package/DBRM accounting section |
| N A6E9BB19... | Beginning of accounting section (DSNDQWAC) |
| O E2C1D56D... | Beginning of distributed section (DSNDQLAC) |
| P 54C4E2D5... | Beginning of MVS/DDF accounting section (DSNDQMDA) |
| Q 0001003A... | Beginning of package/DBRM accounting section (DSNDQPAC) |
| R 00000000... | Beginning of locking accounting section (DSNDQTXA) |
| S 209500E4... | Beginning of SQL accounting section (DSNDQXST) |
| T 00000000... | Beginning of buffer manager accounting section (DSNDQBAC) |
| U 004C011A... | Beginning of product section (DSNDQWHS); beginning of standard header |
| V 004C0200... | Beginning of correlation header (DSNDQWHC) |
| W 00341000... | Beginning of distributed header (DSNDQWHD) |

## Trace field descriptions

If you intend to write a program to read DB2 trace records, use the assembler mapping macros in *prefix*.SDSNMACS.

You can use the TSO or ISPF browse function to look at the field descriptions in the trace record mapping macros online, even when DB2 is down. If you prefer to look at the descriptions in printed form, you can use ISPF to print a listing of the data set.

# Appendix E. Programming for the Instrumentation Facility Interface (IFI)

The information in this appendix is Product-sensitive Programming Interface and Associated Guidance Information as defined in “Notices” on page 1095.

The DB2 instrumentation facility gathers trace data that can be written to one or more destinations that you specify. The instrumentation facility interface (IFI) is designed for a program needing *online* trace information. IFI can be accessed through any of the DB2 attachment facilities.

IFI uses the standard security mechanisms that DB2 uses—connection authorization, plan authorization, and so forth. For more information about security, see “Part 3. Security and auditing” on page 93. Security checks specifically related to IFI are included in the descriptions of the functions.

Before using IFI, you should be familiar with the material in “DB2 trace” on page 1033, which includes information on the DB2 trace facility and instrumentation facility component identifiers (IFCIDs).

Please note that where the trace output indicates a particular release level, you will see 'xx' to show that this information varies according to the actual release of DB2 that you are using.

You can use IFI in a monitor program (a program or function outside of DB2 that receives information about DB2) to perform the following tasks:
- “Submitting DB2 commands through IFI”
- “Obtaining trace data” on page 998
- “Passing data to DB2 through IFI” on page 998

When a DB2 trace is active, internal events trigger the creation of trace records. The records, identified by *instrumentation facility component identifiers* (IFCIDs), can be written to buffers, and you can read them later with the IFI READA function. This means you are collecting the data *asynchronously*; you are not reading the data at the time it was written.

When the DB2 monitor trace is started for class 1, you can trigger the creation of certain types of trace records by an external event—your use of the IFI READS function. The records, identified as usual by IFCIDs, do not need a buffer; they are passed immediately to your monitor program through IFI. This means you are collecting the data *synchronously*. The data is collected at the time of the request for the data.

## Submitting DB2 commands through IFI

You can submit any DB2 command through IFI, but this capability is most useful for submitting DB2 trace commands to start, stop, display, and modify traces.

Using specified trace classes and IFCIDs, a monitor program can control the amount and type of its data. You can design your monitor program to:

- Activate and deactivate pre-defined trace classes.
- Activate and deactivate a trace record or group of records (identified by IFCIDs).

- Activate and deactivate predefined trace classes and trace records (identified by IFCIDs) restricting tracing to a set of DB2 identifiers (plan name, authorization ID, resource manager identifier (RMID), and so on).

## Obtaining trace data

You might want to collect trace data from DB2:

- To obtain accounting information for online billing.
- To periodically obtain system-wide information about DB2, highlight exceptional conditions, or provide throughput information.

  The following illustrates the logic flow:

  1. Initialize.
  2. Set a timer.
  3. Wait for the timer to expire.
  4. Call IFI to obtain statistics data via a READS request.
  5. Do delta calculations to determine activity.

     This step is not necessary for IFCID 0199 because DB2 resets statistics at the beginning of every collection interval.
  6. Display the information on a terminal.
  7. Loop back to the timer.
- To learn which processes have been connected to DB2 the longest, or which processes have used the most CPU time in DB2.
- To obtain accounting records as transactions terminate.
- To determine the access and processing methods for an SQL statement. Start a trace, issue a PREPARE statement, and then use the resulting trace data as an alternative to using EXPLAIN.
- To capture log buffers online for use in remote recovery, as described in “Appendix C. Reading log records” on page 957.
- To retrieve SQL changes synchronously from the log for processing in an application. See “Reading log records with IFI” on page 968 for more information.

## Passing data to DB2 through IFI

You can use IFI to pass data to the destination of a DB2 trace. For example, you can:

- Extend accounting data collected within DB2. For example, a monitor program can collect batch file I/O counts, store them in a user-defined trace record, and process them along with standard DB2 accounting data.
- Include accounting data from QMF, IMS, or CICS.
- Permit CICS users to write the CICS accounting token and task number into the DB2 trace, assuming TOKENE=NO.

## IFI functions

A monitor program can use the following IFI functions:

**COMMAND** To submit DB2 commands. For more information, see “COMMAND: Syntax and usage” on page 1000.

**READS** To obtain monitor trace records synchronously. The READS request causes those records to be returned immediately to the monitor program. For more information, see "READS: Syntax and usage" on page 1002.

**READA** To obtain trace records of any trace type asynchronously. DB2 records trace events as they occur and places that information into a buffer; a READA request moves the buffered data to the monitor program. For more information, see "READA: Syntax and usage" on page 1015.

**WRITE** To write information to a DB2 trace destination that was previously activated by a START TRACE command. For more information, see "WRITE: Syntax and usage" on page 1017.

## Invoking IFI from your program

IFI can be used by assembler and PL/I programs. To use IFI, include a call to DSNWLI in your monitor program.

The following example depicts an IFI call in an assembler program. All examples in this appendix are given for assembler.

```
CALL DSNWLI,(function,ifca,parm-1,...parm-n),VL
```

The parameters passed on the call indicate the function wanted (as described in "IFI functions" on page 998), point to communication areas used by the function, and provide other information that depends on the function specified. Because the parameter list may vary in length, the high-order bit of the last parameter must be on to signal that it is the last parameter in the list. To do this in Assembler for example, use the VL option to signal a variable length parameter list. The communication areas used by IFI are described in "Common communication areas" on page 1019.

After you insert this call in your monitor program, you must link-edit the program with the correct language interface. Each of the following language interface modules has an entry point of DSNWLI for IFI:

CAF DSNALI
CICS DSNCLI
RRSAF DSNRLI
TSO DSNELI
IMS DFSLI000

A second entry point of DSNWLI2 has been added to the CAF (call attachment facility) language interface module, DSNALI. The monitor program that link-edits DSNALI with the program can make IFI calls directly to DSNWLI. The monitor program that loads DSNALI must also load DSNWLI2 and remember its address. When the monitor program calls DSNWLI, the program must have a dummy entry point to handle the call to DSNWLI and then call the real DSNWLI2 routine. See Part 6 of *DB2 Application Programming and SQL Guide* for additional information about using CAF.

***Considerations for writing a monitor program:*** A monitor program issuing IFI requests must be connected to DB2 at the thread level. If the program contains SQL statements, you must precompile the program and create a DB2 plan using the BIND process. If the monitor program does not contain any SQL statements, it does not have to be precompiled. However, as is the case in all the attachment environments, even though an IFI only program (one with no SQL statements) does not have a plan of its own, it can use any plan to get the thread level connection to DB2.

The monitor program can run in either 24- or 31-bit mode.

***Monitor trace classes:*** Monitor trace classes 1 through 8 can be used to collect information related to DB2 resource usage. Use monitor trace class 5, for example, to find out how much time is spent processing IFI requests. Monitor trace classes 2, 3, and 5 are identical to accounting trace classes 2, 3, and 5. For more information about these traces, see “Monitor trace” on page 1036.

***Monitor authorization:*** On the first READA or READS call from a user, an authorization is checked to determine if the primary authorization ID or one of the secondary authorization IDs of the plan executor has MONITOR1 or MONITOR2 privilege. If your installation is using the access control authorization exit routine, then that exit might be controlling the privileges that can use the monitor trace. If you have an authorization failure, an audit trace (class 1) record is generated that contains the return and reason codes from the exit. This is included in IFCID 0140. See “Access control authorization exit” on page 909for more information on the access control authorization exit routine.

## Using IFI from stored procedures

You can use the IFI interface from a stored procedure. The output of the trace can be returned to the client. It is also possible to issue DB2 commands, such as “DISPLAY THREAD”, from a stored procedure and get the results returned to the client.

## COMMAND: Syntax and usage

A DB2 command resides in the output area; a monitor program can submit that command by issuing a COMMAND request to IFI. The DB2 command is processed and the output messages are returned to the monitor program in the return area.

Any DB2 command can be submitted, including START TRACE, STOP TRACE, DISPLAY TRACE, and MODIFY TRACE. Because the program can also issue other DB2 commands, you should be careful about which commands you use. For example, do *not* use STOP DB2.

### Authorization

For an application program to submit a command, the primary authorization ID or one of the secondary authorization IDs of the process must have the appropriate DB2 command authorization, or the request is denied. An application program might have the authorization to issue DB2 commands, but not the authorization to issue READA requests.

### Syntax

```
CALL DSNWLI,('COMMAND ',ifca,return-area,output-area,buffer-info .),VL
```

*ifca* IFCA (instrumentation facility communication area) is an area of storage that contains the return code and reason code indicating the success or failure of the request, diagnostic information from the DB2 component that executed the command, the number of bytes moved to the return area, and the number of bytes of the message segments that did not fit in the return area. It is possible for some commands to complete and return valid information and yet result in the return code and reason code being set to a non-zero value. For example, the DISPLAY DATABASE command may indicate that more information could be returned than was allowed.

If multiple errors occur, the last error is returned to the caller. For example, if the command was in error and the error message did not fit in the area, the error return code and reason code would indicate the return area was too small.

If a monitor program issues START TRACE, the ownership token (IFCAOWNR) in the IFCA determines the owner of the asynchronous buffer. The owner of the buffer is the only process that can obtain data through a subsequent READA request. See “IFCA” on page 1019 for a description of the IFCA.

*return-area*

When the issued command finishes processing, it places messages (if any) in the return area. The messages are stored as varying-length records, and the total number of bytes in the records is placed in the IFCABM (bytes moved) field of the IFCA. If the return area is too small, as many message records as will fit are placed into the return area.

It is the monitor program's responsibility to analyze messages returned by the command function. See “Return area” on page 1022 for a description of the return area.

*output-area*

Contains the varying-length command. See “Output area” on page 1023 for a description of the output area.

*buffer-info*

This parameter is required for starting traces to an OP buffer. Otherwise, it is not needed. This parameter is used only on COMMAND requests. It points to an area containing information about processing options when a trace is started by an IFI call to an unassigned OP*n* destination buffer. An OP*n* destination buffer is considered unassigned if it is not owned by a monitor program.

If the OP*n* destination buffer is assigned, then the buffer information area is not used on a later START or MODIFY TRACE command to that OP*n* destination. For more information about using OP*n* buffers, see “Usage notes” on page 1016.

When you use *buffer-info* on START TRACE, you can specify the number of bytes that can be buffered before the monitor program ECB is posted. The ECB is posted when the amount of trace data collected has reached the value specified in the byte count field. The byte count field is also specified in the buffer information area.

*Table 181. Buffer information area fields. This area is mapped by assembler mapping macro DSNDWBUF.*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| WBUFLEN | 0 | Signed two-byte integer | Length of the buffer information area, plus 4. A zero indicates the area does not exist. |
| | 2 | Signed two-byte integer | Reserved. |
| WBUFEYE | 4 | Character, 4 bytes | Eye catcher for block, WBUF. |

*Table 181. Buffer information area fields (continued). This area is mapped by assembler mapping macro DSNDWBUF.*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| WBUFECB | 8 | Address | The ECB address to post when the buffer has reached the byte count specification (WBUFBC, below). The ECB must reside in monitor key storage. A zero indicates not to post the monitor program. In this case, the monitor program should use its own timer to determine when to issue a READA request. |
| WBUFBC | C | Signed four-byte integer | The records placed into the instrumentation facility must reach this value before the ECB will be posted. If the number is zero, and an ECB exists, posting occurs when the buffer is full. |

## Example

This example issues a DB2 START TRACE command for MONITOR Class 1.

```
CALL DSNWLI,('COMMAND ',IFCAAREA,RETAREA,OUTAREA,BUFAREA),VL
   ⋮

COMMAND DC CL8 'COMMAND '
************************************************************************
* Function parameter declaration                                       *
************************************************************************
* Storage of LENGTH(IFCA) and properly initialized                     *
************************************************************************
IFCAAREA  DS       0CL180
   ⋮

************************************************************************
* Storage for length and returned info.                                *
************************************************************************
RETAREA   DS      CL608
************************************************************************
* Storage for length and DB2 Command                                   *
************************************************************************
OUTAREA   DS      0CL42
OUTLEN    DC      X'002A0000'
OUTCMD    DC      CL38'-STA TRAC(MON) DEST(OPX) BUFSIZE(32)
************************************************************************
* Storage of LENGTH(WBUF) and properly initialized                     *
************************************************************************
BUFAREA  DS       0CL16
   ⋮
```

*Figure 147. Starting a trace using IFI*

## READS: Syntax and usage

READS allows your monitor program to read DB2 status information that is collected at the time of the IFI call. Monitor class 1 must be activated prior to any READS requests. The records available are for IFCIDs 0001, 0002, 0106, 0124, 0129, 0147, 0148, 0149, 0150, 0185, 0199, 0202, 0230, 0254 0306, 0316, and 0317. For a description of the data these records provide, see “Synchronous data” on page 1012. IFCID 0124, 0129, 0147 through 0150, 0197, 0254, 0316, and 0317 can be obtained only through the IFI READS interface.

Monitor class 1 need not be started by the program that issues the READS request, because no ownership of an OP buffer is involved when obtaining data via the

READS interface. Data is written directly to the application program's return area, bypassing the OP buffers. This is in direct contrast to the READA interface where the application that issues READA must first issue a START TRACE command to obtain ownership of an OP buffer and start the appropriate traces.

## Authorization

On a READS request, a check is made to see if monitor class 1 is active; if it is not active, the request is denied. The primary authorization ID or one of the secondary authorization IDs of the process running the application program must have MONITOR1 or MONITOR2 privilege. If neither the primary authorization ID nor one of the secondary authorization IDs has authorization, the request is denied. IFCID 185 requests are an exception—they do not require the MONITOR1 or MONITOR2 privilege. READS requests are checked for authorization once for each user (ownership token) of the thread. (Several users can use the same thread, but an authorization check is performed each time the user of the thread changes.)

If you use READS to obtain your own data (IFCID 0124, 0147, 0148, or 0150 not qualified), then no authorization check is performed.

## Syntax

```
CALL DSNWLI,('READS   ',ifca,return-area,ifcid-area,qual-area),VL
```

*ifca*
Contains information about the success of the call. See “IFCA” on page 1019 for a description of the IFCA.

*return-area*
Contains the varying-length records returned by the instrumentation facility. IFI monitor programs might need large enough READS return areas to accommodate the following:

- Larger IFCID 0147 and 0148 records containing distributed thread data (both allied and database access) that is returned to them.
- Additional records returned when database access threads exist that satisfy the specified qualifications on the READS request.
- Log record control intervals with IFCID 129. For more information about using IFI to return log records, see “Reading specific log records (IFCID 0129)” on page 968.
- Log records based on user-specified criteria with IFCID 306. For example, the user can retrieve compressed or decompressed log records. For more information about reading log records, see “Appendix C. Reading log records” on page 957.
- Data descriptions and changed data returned with IFCID 185.

If the return area is too small to hold all the records returned, it contains as many records as will fit. The monitor program obtains the return area for READS requests in its private address space. See “Return area” on page 1022 for a description of the return area.

*ifcid-area*
Contains the IFCIDs of the information wanted. The number of IFCIDs can be variable. If the length specification of the IFCID area is exceeded or an IFCID of X’FFFF’ is encountered, the list is terminated. If an invalid IFCID is specified no data is retrieved. See “IFCID area” on page 1023 for a description of the IFCID area.

*qual-area*

This parameter is optional, and is used only on READS requests. It points to the qualification area, where a monitor program can specify constraints on the data that is to be returned. If the qualification area does not exist (length of binary zero), information is obtained from all active allied threads and database access threads. Information is not obtained for any inactive database access threads that might exist.

The length constants for the qualification area are provided in the DSNDWQAL mapping macro. If the length is not equal to the value of one of these constants, IFI considers the call invalid.

The following trace records, identified by IFCID, cannot be qualified; if you attempt to qualify them, the qualification is ignored: 0001, 0002, 0106, 0202, 0230.

The rest of the synchronous records can be qualified. See “Synchronous data” on page 1012 for information about these records. However, not all the qualifications in the qualification area can be used for these IFCIDs. See “Which qualifications are used?” on page 1010 for qualification restrictions. Unless the qualification area has a length of binary zero (in which case the area does not exist), the address of *qual-area* supplied by the monitor program points to an area formatted by the monitor program as shown in Table 182.

*Table 182. Qualification area fields. This area is mapped by the assembler mapping macro DSNDWQAL.*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| WQALLEN | 0 | Signed two-byte integer | Length of the qualification area, plus 4. The following constants set the qualification area length field:<br>• WQALLN21. When specified, the location name qualifications (WQALLOCN and WQALLUWI) are ignored.<br>• WQALLN22. When specified, the location name qualifications (WQALLOCN and WQALLUWI) are used.<br>• WQALLN23. When specified, the log data access fields (WQALLTYP, WQALLMOD, WQALLRBA, and WQALLNUM) are used for READS calls using IFCID 129.<br>• WQALLN4. When specified, the location name qualifications (WQALLOCN and WQALLUWI), the group buffer pool qualifier (WQALGBPN) and the read log fields are used.<br>• WQALLN5. When specified, the dynamic statement cache fields (WQALFFLD, WQALFVAL, WQALSTNM, and WQALSTID) are used for READS calls for IFCID 0316 and 0317.<br>• WQALLN6. When specified, the end-user identification fields (WQALEUID, WQALEUTX, and WQALEUWS) are used for READS calls for IFCID 0124, 0147, 0148, 0149, and 0150. |
| | 2 | Signed two-byte integer | Reserved. |
| WQALEYE | 4 | Character, 4 bytes | Eye catcher for block, WQAL. |
| WQALACE | 8 | Address | Thread identification token value. This value indicates the specific thread wanted; binary zero if it is not to be used. |
| WQALAIT2 | C | Address | Reserved. |
| WQALPLAN | 10 | Character, 8 bytes | Plan name; binary zero if it is not to be used. |
| WQALAUTH | 18 | Character, 8 bytes | The current primary authorization ID; binary zero if it is not to be used. |
| WQALOPID | 20 | Character, 8 bytes | The original authorization ID; binary zero if it is not to be used. |
| WQALCONN | 28 | Character, 8 bytes | Connection name; binary zero if it is not to be used. |

*Table 182. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| WQALCORR | 30 | Character, 12 bytes | Correlation ID; binary zero if it is not to be used. |
| WQALREST | 3C | Character, 32 bytes | Resource token for a specific lock request when IFCID 0149 is specified. The field must be set by the monitor program. The monitor program can obtain the information from a previous READS request for IFCID 0150 or from a READS request for IFCID 0147 or 0148. |
| WQALHASH | 5C | Hex, 4 bytes | Resource hash value specifying the resource token for a specific lock request when IFCID 0149 is specified. The field must be set by the monitor program. The monitor program can obtain the information from a previous READS request for IFCID 0150 or possibly from a READS request for IFCID 0147 or 0148. |
| WQALASID | 60 | Hex, 2 bytes | ASID specifying the address space of the process wanted. |
| WQALFOPT | 62 | Hex, 1 byte | Filtering options for IFCID 0150:<br>• X'80' - Return lock information only for resources that have waiters.<br>• X'40' - Return lock information only for resources that have one or more interested agents. |
|  | 63 | Hex, 1 byte | Reserved. |
|  | 64 | Character, 24 bytes | LUWID (logical unit of work ID) of the thread wanted; binary zero if it is not to be used |
|  | 7C | Character, 16 bytes | Location name. If specified, then data is returned only for distributed agents, which originate at the specified location. For example, if site A is located where the IFI program is running and SITE A is specified in the WQALLOCN, then distributed agents, both database access threads and distributed allied agents, executing at SITE A are reported. Local non-distributed agents are not reported. If site B is specified and the IFI program is still executing at site A, then information on database access threads which are executing in support of a distributed allied agent at site B are reported. If WQALLOCN is not specified, then information on all threads executing at SITE A (the site where the IFI program is executing) is returned. This includes local non-distributed threads, local database access agents, and local distributed allied agents. |
| WQALLTYP | 8C | Character, 3 bytes | Specifies the type of log data access. 'CI ' must be specified to obtain log record control intervals (CIs). |

*Table 182. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| WQALLMOD | 8F | Character, 1 byte | The mode of log data access:<br>• 'D' - return the direct log record specified in WQALLRBA if the IFCID is 0306.<br>• 'F' - access the first log CI of the restarted DB2 system if the IFCID is 0129. One CI is returned, and the WQALLNUM and WQALLRBA fields are ignored. It indicates to return the first set of qualified log records if the IFCID is 0306.<br>• 'R' - access the CIs specified by the value in the WQALLRBA field:<br>– If the requested number of complete CIs (as specified in WQALLNUM) are currently available, those CIs are returned. If fewer than the requested number of complete CIs are available, IFI returns as many complete CIs as are available.<br>– If the WQALLRBA value is beyond the end of the active log, IFI returns a return code of X'0000000C' and a reason code of X'00E60855'. No records are returned.<br>– If no complete CIs exist beyond the WQALLRBA value, IFI returns a return code of X'0000000C' and a reason code of X'00E60856'. No records are returned.<br>• 'H' - return the highest LRSN or log RBA in the active log. The value is returned in the field IFCAHLRS in the IFCA.<br>• 'N' - return the next set of qualified log records.<br>• 'T' - terminate the log position that is held to anticipate a future mode 'N' call.<br>• 'P' - the last partial CI written to the active log is given to the Log Capture Exit. If the last CI written to the log was not full, the RBA of the log CI given to the Log Exit is returned in the IFCAHLRS field of the IFI communication area (IFCA). Otherwise, an RBA of zero is returned in IFCAHLRS. This option ignores WQALLRBA and WQALLNUM. |
| WQALLNUM | 90 | Hex, 2 bytes | The number of log CIs to be returned. The valid range is X'0001' to X'0007'. |
| WQALCDCD | 92 | Character, 1 byte | Data description request flag (A,Y,N):<br>• 'A' indicates that a data description will only be returned the first time a DATA request is issued from the region or when it was changed for a given table. This is the default.<br>• 'Y' indicates that a data description will be returned for each table in the list for every new request.<br>• 'N' indicates that a data description will not be returned. |
|  | 93 | Hex, 1 byte | Reserved. |
| WQALLRBA | 94 | Hex, 8 bytes | • If the IFCID is 0129, the starting log RBA of the CI to be returned. The CI starting log RBA value must end in X'000'. The RBA value must be right-justified.<br>• If the IFCID is 0306, this is the log RBA or LRSN to be used in mode 'F'. |

*Table 182. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| WQALGBPN | 9C | Character, 8 bytes | Group buffer pool name for IFCID 0254. Buffer pool name for IFCID 0199. To specify a single buffer pool or group buffer pool, specify the buffer pool name in hexadecimal, followed by hexadecimal blanks. For example, to specify buffer pool BP1, put X'C2D7F14040404040' in this field. To specify more than one buffer pool or group buffer pool, use the pattern-matching character X'00' in any position in the buffer pool name. X'00' indicates that any character can appear in that position, and in all positions that follow. For example, if you put X'C2D7F10000000000' in this field, you indicate that you want data for all buffer pools whose names begin with BP1, so IFI collects data for BP1, BP10 through BP19, and BP16K0 through BP16K9. If you put X'C2D700F100000000' in this field, you indicate that you want data for all buffer pools whose names begin with BP, so IFI collects data for all buffer pools. IFI ignores X'F1' in position four because it occurs after the first X'00'. |
| WQALLCRI | A4 | Hex, 1 byte | Log Record Selection Criteria<br>• '00' indicates the return DB2CDC and UR control log records. |
| WQALLOPT | A5 | Hex, 1 byte | Processing Options relating to decompression<br>• '01' indicates to decompress the log records if they are compressed.<br>• '00' indicates that decompression should not occur. |

*Table 182. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| WQALFLTR | A6 | Hex, 1 byte | For an IFCID 0316 request, identifies the filter method:<br>• X'00' indicates no filtering. This value tells DB2 to return information for as many cached statements as fit in the return area.<br>• X'01' indicates that DB2 returns information about the cached statements that have the highest values for a particular statistics field. Specify the statistics field in WQALFFLD. DB2 returns information for as many statements as fit in the return area. For example, if the return is large enough for information about 10 statements, the statements with the ten highest values for the specified statistics field are reported.<br>• X'02' indicates that DB2 returns information about the cached statements that exceed a threshold value for a particular statistics field. Specify the name of the statistics field in WQALFFLD. Specify the threshold value in WQALFVAL. DB2 returns information for as many qualifying statements as fit in the return area.<br>• X'04' indicates that DB2 returns information about a single cached statement. The application provides the four-byte cached statement identifier in field WQALSTID. An IFCID 0316 request with this qualifier is intended for use with IFCID 0172 or IFCID 0196, to obtain information about the statements that are involved in a timeout or deadlock.<br>For an IFCID 0317 request, identifies the filter method:<br>• X'04' indicates that DB2 returns information about a single cached statement. The application provides the four-byte cached statement identifier in field WQALSTID. An IFCID 0317 request with this qualifier is intended for use with IFCID 0172 or IFCID 0196, to obtain information about the statements that are involved in a timeout or deadlock.<br>For an IFCID 0306 request, indicates whether DB2 merges log records in a data sharing environment:<br>• X'00' indicates that DB2 merges log records from data sharing members.<br>• X'03' indicates that DB2 does not merge log records from data sharing members. |

*Table 182. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| WQALFFLD | A7 | Character, 1 byte | For an IFCID 0316 request, when WQALFLTR is X'01' or X'02', this field specifies the statistics field used to determine the cached statements about which DB2 reports. The following list shows the values you can enter and the statistics fields they represent:<br>• ’E’ - the number of executions of the statement (QW0316NE)<br>• ’B’ - the number of buffer reads (QW0316NB)<br>• ’G’ - the number of GETPAGE requests (QW0316NB)<br>• ’R’ - the number of rows examined (QW0316NR)<br>• ’P’ - the number of rows processed (QW0316NP)<br>• ’S’ - the number of sorts performed (QW0316NS)<br>• ’I’ - the number of index scans (QW0316NI)<br>• ’T’ - the number of table space scans (QW0316NT)<br>• ’L’ - the number of parallel groups (QW0316NL)<br>• ’W’ - the number of buffer writes (QW0316NW)<br>• ’A’ - the accumulated elapsed time (QW0316AE). This option is valid only when QWALFLTR=X'01'.<br>• ’X’ - the number of times that a RID list was not used because the number of RIDs would have exceeded one or more internal DB2 limits (QW0316RT).<br>• ’Y’ - the number of times that a RID list was not used because not enough storage was available (QW0316RS).<br>• ’C’ - the accumulated CPU time (QW0316CT). This option is valid only when QWALFLTR=X'01'.<br>• ’1’ - the accumulated wait time for synchronous I/O (QW0316W1). This option is valid only when QWALFLTR=X'01'.<br>• ’2’ - the accumulated wait time for lock and latch requests (QW0316W2). This option is valid only when QWALFLTR=X'01'.<br>• ’3’ - the accumulated wait time for a synchronous execution unit switch (QW0316W3). This option is valid only when QWALFLTR=X'01'.<br>• ’4’ - the accumulated wait time for global locks (QW0316W4). This option is valid only when QWALFLTR=X'01'.<br>• ’5’ - the accumulated wait time for read activity by another thread (QW0316W5). This option is valid only when QWALFLTR=X'01'.<br>• ’6’ - the accumulated wait time for write activity by another thread (QW0316W6). This option is valid only when QWALFLTR=X'01'. |
| WQALFVAL | A8 | Signed 4-byte integer | For an IFCID 0316 request, when WQALFLTR is X'02', this field and WQALFFLD determine the cached statements about which DB2 reports.<br>To be eligible for reporting, a cached statement must have a value for the statistics field specified by WQALFFLD that is no smaller than the value you specify in this field. DB2 reports information on as many eligible statements as fit in the return area. |
| WQALSTNM | AC | Character, 16 bytes | For an IFCID 0317 request, when WQALFLTR is not X'04', this field specifies the name of a cached statement about which DB2 reports. This is a name that DB2 generates when it caches the statement. To obtain this name, issue a READS request for IFCID 0316. The name is in field QW0316NM. This field and WQALSTID uniquely identify a cached statement. |

*Table 182. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| WQALSTID | BC | Unsigned 4-byte integer | For an IFCID 0316 or IFCID 0317 request, this field specifies the ID of a cached statement about which DB2 reports. This is an ID that DB2 generates when it caches the statement.<br>• For an IFCID 0317 request, when WQALFLTR is not X'04', obtain this ID by issuing a READS request for IFCID 0316. The ID is in field QW0316TK. This field and WQALSTNM uniquely identify a cached statement.<br>• For an IFCID 0316 or IFCID 0317 request, when WQALFLTR is X'04', obtain this ID by issuing a READS request for IFCID 0172 or IFCID 0196. The ID is in field QW0172H9 (cached statement ID for the holder in a deadlock), QW0172W9 (cached statement ID for the waiter in a deadlock), or QW0196H9 (cached statement ID of the holder in a timeout). This field uniquely identifies a cached statement. |
| WQALEUID | C0 | Character, 16 bytes | The end user's workstation user ID. This value can be different from the authorization ID used to connect to DB2. This field contains binary zeroes if the client did not supply this information. |
| WQALEUTX | D0 | Character, 32 bytes | The name of the transaction or application that the end user is running. This value identifies the application that is currently running, not the product that is used to run the application. This field contains binary zeroes if the client did not supply this information. |
| WQALEUWS | F0 | Character, 18 bytes | The end user's workstation name. This value can be different from the authorization ID used to connect to DB2. This field contains binary zeroes if the client did not supply this information. |

**Note:** If your monitor program does not initialize the qualification area, the READS request is denied.

## Which qualifications are used?

Not all qualifications are used for all IFCIDs. The following table lists the qualification fields that are used for each IFCID.

*Table 183. Qualification fields for IFCIDs*

| These IFCIDs... | Are allowed to use these qualification fields |
|---|---|
| 0124, 0147, 0148, 0150 | WQALACE<br>WQALAIT2<br>WQALPLAN[1]<br>WQALAUTH[1]<br>WQALOPID[1]<br>WQALCONN[1]<br>WQALCORR[1]<br>WQALASID<br>WQALLUWI[1]<br>WQALLOCN[1]<br>WQALEUID<br>WQALEUTX<br>WQALEUWS |
| 0129 | WQALLTYP<br>WQALLMOD<br>WQALLRBA<br>WQALLNUM |

*Table 183. Qualification fields for IFCIDs (continued)*

| These IFCIDs... | Are allowed to use these qualification fields |
|---|---|
| 0149 | WQALREST<br>WQALHASH |
| 0150 | WQALFOPT |
| 0185 | WQALCDCD |
| 0199, 0254 | WQALGBPN[2] |
| 0306 | WQALFLTR<br>WQALLMOD<br>WQALLRBA<br>WQALLCRI<br>WQALLOPT |
| 0316 | WQALFLTR<br>WQALFFLD<br>WQALFVAL<br>WQALSTID |
| 0317 | WQALFLTR<br>WQALSTNM<br>WQALSTID |

**Note:**

1. DB2 allows you to partially qualify a field and fill the rest of the field with binary zero. For example, the 12-byte correlation value for a CICS thread contains the 4-character CICS transaction code in positions 5-8. Assuming a CICS transaction code of AAAA, the following hexadecimal *qual-area* correlation qualification can be used to find the first transaction with a correlation value of AAAA in positions 5-8: X'00000000C1C1C1C100000000'.
2. X'00' in this field indicates a pattern-matching character. X'00' in any position of the field indicates that IFI collects data for buffer pools whose names contain any character in that position and all following positions.

## Usage notes

Due to performance considerations, the majority of data obtained by a monitor program probably comes over the synchronous interface: summarized DB2 information is easier for a monitor program to process, and the monitor program logic is simpler since a smaller number of records are processed.

After you issue the START TRACE command to activate monitor class 1, you can issue a READS request to obtain information immediately and return the information to your monitor program in the return area. Start monitor classes 2, 3, 5, 7, and 8 to collect additional summary and status information for later probing. In this case an instrumentation facility trace is started and information is summarized by the instrumentation facility, but not returned to the caller until requested by a READS call.

The READS request may reference data being updated during the retrieval process. It might be necessary to do reasonability tests on data obtained through READS. The READS function does not suspend activity taking place under structures being referred to. Thus, an abend can occur. If so, the READS function is terminated without a dump and the monitor program is notified through the return code and reason code information in the IFCA. However, the return area can contain valid trace records, even if an abend occurred; therefore, your monitor program should check for a non-zero value in the IFCABM (bytes moved) field of the IFCA.

When using a READS with a query parallelism task, keep in mind that each parallel task is a separate thread. Each parallel thread has a separate READS output. See “Chapter 34. Parallel operations and query performance” on page 841 for more information on tracing the parallel tasks. It is also possible that a READS request might return thread information for parallel tasks on a DB2 data sharing member without the thread information for the originating task in a Sysplex query parallelism case. See *DB2 Data Sharing: Planning and Administration* .

When starting monitor trace class 1, specifying a PLAN, an AUTHID, an RMID, or a LOCATION has no effect on the number of records returned on IFI READS requests. The qual-area parameter, mapped by DSNDWQAL, is the only means of qualifying the trace records to be returned on IFI READS requests.

## Synchronous data

There are certain types of records that you can read synchronously, as long as monitor trace class 1 is active. Identified by IFCID, these records are:

**0001** Statistical data on the systems services address space, including task control block (TCB) and service request block (SRB) times for system services, database services, including DDF statistics, and Internal Resource Lock Manager (IRLM) address spaces.

**0002** Statistical data on the database services address space.

**0106** Static system parameters.

**0124** An active SQL snapshot giving status information about the process, the SQL statement text, the relational data system input parameter list (RDI) block, and status flags to indicate certain bind and locking information.

It is possible to obtain a varying amount of data because the request requires the process to be connected to DB2, have a cursor table allocated (RDI and status information is provided), and be active in DB2 (SQL text is provided if available). The SQL text provided does not include the SQL host variables.

For dynamic SQL, IFI provides the original SQL statement. The RDISTYPE field contains the actual SQL function taking place. For example, for a SELECT statement, the RDISTYPE field can indicate that an open cursor, fetch, or other function occurred. For static SQL, you can see the DECLARE CURSOR statement, and the RDISTYPE indicates the function. The RDISTYPE field is mapped by mapping macro DSNXRDI.

**0129** Returns one or more VSAM control intervals (CIs) containing DB2 recovery log records. For more information about using IFI to return these records for use in remote site recovery, see “Appendix C. Reading log records” on page 957.

**0147** An active thread snapshot giving a status summary of processes at a DB2 thread or non-thread level.

**0148** An active thread snapshot giving more detailed status of processes at a DB2 thread or non-thread level.

**0149** Information indicating who (the thread identification token) is holding locks and waiting for locks on a particular resource and hash token. The data provided is in the same format defined for IFCID 0150.

**0150** All the locks held and waited on by a given user or owner (thread identification token).

**0199** Information about buffer pool usage by DB2 data sets. DB2 reports this information for an interval you specify in field DATASET STATS TIME of installation panel DSNTIPN. At the beginning of each interval, DB2 resets these statistics to 0.

**0202** Dynamic system parameters.

**0230** Global statistics for data sharing.

**0254** Group buffer pool usage in the data sharing group.

**0316** Returns information about the contents of the dynamic statement cache. The IFI application can request information for all statements in the cache, or provide qualification parameters to limit the data returned. DB2 reports the following information about a cached statement:

- A statement name and ID that uniquely identify the statement
- If IFCID 0318 is active, performance statistics for the statement
- The first 60 bytes of the statement text

**0317** Returns the complete text of an SQL statement in the dynamic statement cache. To identify a statement for which you want the complete text, you must the statement name and statement ID from IFCID 0316 output. For more information on using IFI to obtain information about the dynamic statement cache, see “Using READS calls to monitor the dynamic statement cache”.

You can read another type of record synchronously as long as monitor trace class 6 is active:

**0185** Data descriptions for each table for which captured data is returned on this DATA request. IFCID 0185 data is only available through a propagation exit routine triggered by DB2.

**0306** Returns compressed or decompressed log records in both a data sharing or non data-sharing environment. For IFCID 306 requests, your program's return area must reside in ECSA key 7 storage with the IFI application program running in key 0 supervisor state. The IFI application program must set the eye-catcher to “I306” before making the IFCID 306 call. See “IFCA” on page 1019 for more information on the instrumentation facility communication area (IFCA) and what is expected of the monitor program.

For more information on IFCID field descriptions, see the mapping macros in *prefix*.SDSNMACS. See also “DB2 trace” on page 1033 and “Appendix D. Interpreting DB2 trace output” on page 981 for additional information.

## Using READS calls to monitor the dynamic statement cache

You can use READS requests from an IFI application to monitor the contents of the dynamic statement cache, and optionally, to see some accumulated statistics for those statements. This can help you detect and diagnose performance problems for those cached dynamic SQL statements.

An IFI program that monitors the dynamic statement cache should include these steps:

1. Acquire and initialize storage areas for common IFI communication areas.
2. Issue an IFI COMMAND call to start monitor trace class 1.

   This lets you make READS calls for IFCID 0316 and IFCID 0317.
3. Issue an IFI COMMAND call to start performance trace class 30 for IFCID 0318.

   This enables statistics collection for statements in the dynamic statement cache. See “Controlling collection of dynamic statement cache statistics with IFCID 0318” on page 1015 for information on when you should start a trace for IFCID 0318.
4. Put the IFI program into a wait state.

   During this time, SQL applications in the subsystem execute dynamic SQL statements using the dynamic statement cache.
5. Resume the IFI program after enough time has elapsed for a reasonable amount of activity to occur in the dynamic statement cache.
6. Set up the qualification area for a READS call for IFCID 0316 as described in Table 182 on page 1004.
7. Set up the IFCID area to request data for IFCID 0316.
8. Issue an IFI READS call to retrieve the qualifying cached SQL statements.
9. Examine the contents of the return area.

   For a statement with unexpected statistics values:

   a. Obtain the statement name and statement ID from the IFCID 0316 data.

   b. Set up the qualification area for a READS call for IFCID 0317 as described in Table 182 on page 1004.

   c. Set up the IFCID area to request data for IFCID 0317.

   d. Issue a READS call for IFCID 0317 to get the entire text of the statement.

   e. Obtain the statement text from the return area.

   f. Use the statement text to execute an SQL EXPLAIN statement.

   g. Fetch the EXPLAIN results from the PLAN_TABLE.
10. Issue an IFI COMMAND call to stop monitor trace class 1.
11. Issue an IFI COMMAND call to stop performance trace class 30 for IFCID 0318.

An IFI program that monitors deadlocks and timeouts of cached statements should include these steps:

1. Acquire and initialize storage areas for common IFI communication areas.
2. Issue an IFI COMMAND call to start monitor trace class 1.

   This lets you make READS calls for IFCID 0316 and IFCID 0317.
3. Issue an IFI COMMAND call to start performance trace class 30 for IFCID 0318.

   This enables statistics collection for statements in the dynamic statement cache. See “Controlling collection of dynamic statement cache statistics with IFCID 0318” on page 1015 for information on when you should start a trace for IFCID 0318.
4. Start performance trace class 3 for IFCID 0172 to monitor deadlocks, or performance trace class 3 for IFCID 0196 to monitor timeouts.
5. Put the IFI program into a wait state.

   During this time, SQL applications in the subsystem execute dynamic SQL statements using the dynamic statement cache.

6. Resume the IFI program when a deadlock or timeout occurs.
7. Issue a READA request to obtain IFCID 0172 or IFCID 0196 trace data.
8. Obtain the cached statement ID of the statement that was involved in the deadlock or timeout from the IFCID 0172 or IFCID 0196 trace data. Using the statement ID, set up the qualification area for a READS call for IFCID 0316 or IFCID 0317 as described in Table 182 on page 1004.
9. Set up the IFCID area to request data for IFCID 0316 or IFCID 0317.
10. Issue an IFI READS call to retrieve the qualifying cached SQL statement.
11. Examine the contents of the return area.
12. Issue an IFI COMMAND call to stop monitor trace class 1.
13. Issue an IFI COMMAND call to stop performance trace class 30 for IFCID 0318 and performance trace class 3 for IFCID 0172 or IFCID 0196.

## Controlling collection of dynamic statement cache statistics with IFCID 0318

The collection of statistics for statements in the dynamic statement cache can increase the processing cost for those statements. To minimize this increase, use IFCID 0318 to enable and disable the collection of dynamic statement cache statistics. When IFCID 0318 is inactive, DB2 does not collect those statistics. DB2 tracks the statements in the dynamic statement cache, but does not accumulate the statistics as those statements are used. When you are not actively monitoring the cache, you should turn off the trace for IFCID 0318.

If you issue a READS call for IFCID 0316 while IFCID 0318 is inactive, DB2 returns identifying information for all statements in the cache, but returns 0 in all the IFCID 0316 statistics counters.

When you stop or start the trace for IFCID 0318, DB2 resets the IFCID 0316 statistics counters for all statements in the cache to 0.

# READA: Syntax and usage

The READA function allows a monitor program to asynchronously read data that has accumulated in an OP*n* buffer.

## Authorization

On a READA request the application program must own the specified destination buffer, or the request is denied. You can obtain ownership of a storage buffer by issuing a START TRACE to an OP*n* destination. The primary authorization ID or one of the secondary authorization IDs of the process must have MONITOR1 or MONITOR2 privilege or the request is denied. READA requests are checked for authorization once for each user of the thread. (Several users can use the same thread, but an authorization check is performed each time the user of the thread changes.)

## Syntax

```
CALL DSNWLI,('READA   ',ifca,return-area),VL
```

*ifca*

Contains information about the OP*n* destination and the ownership token value (IFCAOWNR) at call initiation. After the READA call has been completed, the IFCA contains the return code, reason code, the number of bytes moved to the return area, the number of bytes not moved to the return area if the area was too small, and the number of records lost. See “Common communication areas” on page 1019 for a description of the IFCA.

*return-area*

Contains the varying-length records returned by the instrumentation facility. If the return area is too small, as much of the output as will fit is placed into the area (a complete varying-length record). Reason code 00E60802 is returned in cases where the monitor program's return area is not large enough to hold the returned data. See “Return area” on page 1022 for a description of the return area.

IFI allocates up to 8 OP buffers upon request from storage above the line in extended CSA. IFI uses these buffers to store trace data until the owning application performs a READA request to transfer the data from the OP buffer to the application's return area. An application becomes the owner of an OP buffer when it issues a START TRACE command and specifies a destination of OPN or OPX. Each buffer can be of size 4K to 1M. IFI allocates a maximum of 4MB of storage for the 8 OP buffers. The default monitor buffer size is determined by the MONSIZE parameter in the DSNZPARM module.

## Usage notes

You can use a monitor trace that uses any one of eight online performance monitor destinations, OP*n*, (where *n* is equal to a value from 1 to 8). Typically, the destination of OP*n* is only used with commands issued from a monitor program. For example, the monitor program can pass a specific online performance monitor destination (OP1, for example) on the START TRACE command to start asynchronous trace data collection.

If the monitor program passes a generic destination of OPX, the instrumentation facility assigns the next available buffer destination slot and returns the OP*n* destination name to the monitor program. To avoid conflict with another trace or program that might be using an OP buffer, you should use the generic OPX specification when you start tracing. You can then direct the data to the destination specified by the instrumentation facility with the START or MODIFY TRACE commands.

There are times, however, when you should use a specific OP*n* destination initially:

- When you plan to start numerous asynchronous traces to the same OP*n* destination. To do this, you must specify the OP*n* destination in your monitor program. The OP*n* destination started is returned in the IFCA.
- When the monitor program specifies that a particular monitor class (defined as available) together with a particular destination (for example OP7) indicates that certain IFCIDs are started. An operator can use the DISPLAY TRACE command to determine which monitors are active and what events are being traced.

***Buffering data:*** To have trace data go to the OP*n* buffer, you must start the trace from within the monitor program. After the trace is started, DB2 collects and buffers the information as it occurs. The monitor program can then issue a read asynchronous (READA) request to move the buffered data to the monitor program. The buffering technique ensures that the data is not being updated by other users while the buffer is being read by the READA caller. For more information, see “Data integrity” on page 1027.

***Possible data loss:*** Although it is possible to activate all traces and have the trace data buffered, it is definitely not recommended, because performance might suffer and data might be lost.

Data loss occurs when the buffer fills before the monitor program can obtain the data. DB2 does not wait for the buffer to be emptied, but, instead, informs the monitor program on the next READA request (in the IFCARLC field of the IFCA) that the data has been lost. It is the user's responsibility to have a high enough dispatching priority that the application can be posted and then issue the READA request before significant data is lost.

## Asynchronous data

DB2 buffers all IFCID data that is activated by the START TRACE command and passes it to a monitor program on a READA request. The IFCID events include all of the following:

- Serviceability
- Statistics
- Accounting
- Performance
- Audit data
- IFCIDs defined for the IFI write function

IFCID events are discussed in "DB2 trace" on page 1033.

Your monitor program can request an asynchronous buffer, which records trace data as trace events occur. The monitor program is then responsible for unloading the buffer on a timely basis. One method is to set a timer to wake up and process the data. Another method is to use the buffer information area on a START TRACE command request, shown in Table 181 on page 1001, to specify an ECB address to post when a specified number of bytes have been buffered.

## Example

The following depicts the logic flow for monitoring DB2 accounting and for displaying the information on a terminal:

1. Initialize.
2. Use GETMAIN to obtain a storage area equal to BUFSIZE.
3. Start an accounting trace by issuing a DB2 START TRACE=ACCTG DEST=OPX command through IFI indicating to wake up this routine by a POST whenever the buffer is 20% full.
4. Check the status in the IFCA to determine if the command request was successful.
5. WAIT for the buffer to be posted.
6. Clear the post flag.
7. Call IFI to obtain the buffer data via a READA request.
8. Check the status of the IFCA to determine if the READA request was successful.
9. De-block the information provided.
10. Display the information on a terminal.
11. Loop back to the WAIT.

# WRITE: Syntax and usage

A monitor program can write information to a DB2 trace destination by issuing a write (WRITE) request for a specific IFCID.

## Authorization

WRITE requests are not checked for authorization, but a DB2 trace must be active for the IFCID being written. If the IFCID is not active, the request is denied. For a WRITE request, no other authorization checks are made.

## Syntax

```
CALL DSNWLI,('WRITE   ',ifca,output-area,ifcid-area),VL
```

The write function must specify an IFCID area. The data written is defined and interpreted by your site.

*ifca* Contains information regarding the success of the call. See “IFCA” on page 1019 for a description of the IFCA.

*output-area*
Contains the varying-length of the monitor program's data record to be written. See “Output area” on page 1023 for a description of the output area.

*ifcid-area*
Contains the IFCID of the record to be written. Only the IFCIDs defined to the write function (see Table 184) are allowed. If an invalid IFCID is specified or the IFCID is not active (was not started by a TRACE command), no data is written. See Table 184 for IFCIDs that can be used by the write function.

*Table 184. Valid IFCIDs for WRITE Function*

| IFCID (decimal) | IFCID (hex) | Trace type | Class | Comment |
|---|---|---|---|---|
| 0146 | 0092 | Auditing | 9 | Write to IFCID 146 |
| 0151 | 0097 | Accounting | 4 | Write to IFCID 151 |
| 0152 | 0098 | Statistics | 2 | Write to IFCID 152 |
| 0153 | 0099 | Performance | 1 | Background events and write to IFCID 153 |
| 0154 | 009A | Performance | 15 | Write to IFCID 154 |
| 0155 | 009B | Monitoring | 4 | Write to IFCID 155 |
| 0156 | 009C | Serviceability | 6 | Reserved for user-defined serviceability trace |

See “IFCID area” on page 1023 for a description of the IFCID area.

## Usage notes

The information is written to the destination that was previously activated by a START TRACE command for that ID.

If your site uses the IFI write function, you should establish usage procedures and standards. Procedures are necessary to ensure that the correct IFCIDs are active when DB2 is performing the WRITE function. Standards are needed to determine what records and record formats a monitor program should send to DB2. You should place your site's record type and sub-type in the first fields in the data record since your site can use one IFCID to contain many different records.

## Common communication areas

The following communication areas are used on all IFI calls:
- “IFCA”, below
- “Return area” on page 1022
- “IFCID area” on page 1023
- “Output area” on page 1023

## IFCA

The program's IFCA (instrumentation facility communication area) is a communications area between the monitor program and IFI. A required parameter on all IFI requests, the IFCA contains information about the success of the call in its return code and reason code fields.

*The monitor program is responsible for allocating storage for the IFCA and initializing it.* The IFCA must be initialized to binary zeros and the eye catcher, 4-byte owner field, and length field must be set by the monitor program. Failure to properly initialize the IFCA results in denying any IFI requests.

The monitor program is also responsible for checking the IFCA return code and reason code fields to determine the status of the request.

The IFCA fields are described in Table 185.

*Table 185. Instrumentation facility communication area. The IFCA is mapped by assembler mapping macro DSNDIFCA.*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| IFCALEN | 0 | Hex, 2 bytes | Length of IFCA. |
| IFCAFLGS | 2 | Hex, 1 byte | Processing flags.<br>• IFCAGLBL, X'80'<br>This bit is on if an IFI request is to be processed on all members of a data sharing group. |
|  | 3 | Hex, 1 byte | Reserved. |
| IFCAID | 4 | Character, 4 bytes | Eye catcher for block, IFCA. |
| IFCAOWNR | 8 | Character, 4 bytes | Owner field, provided by the monitor program. This value is used to establish ownership of an OP*n* destination and to verify that a requester can obtain data from the OP*n* destination. This is *not* the same as the owner ID of a plan. |
| IFCARC1 | C | Four-byte signed integer | Return code for the IFI call. Binary zero indicates a successful call. See Part 3 of *DB2 Messages and Codes* for information about reason codes. For a return code of 8 from a COMMAND request, the IFCAR0 and IFCAR15 values contain more information. |
| IFCARC2 | 10 | Four-byte signed integer | Reason code for the IFI call. Binary zero indicates a successful call. See Part 3 of *DB2 Messages and Codes* for information about reason codes. |
| IFCABM | 14 | Four-byte signed integer | Number of bytes moved to the return area. A non-zero value in this field indicates information was returned from the call. Only complete records are moved to the monitor program area. |

*Table 185. Instrumentation facility communication area (continued). The IFCA is mapped by assembler mapping macro DSNDIFCA.*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| IFCABNM | 18 | Four-byte signed integer | Number of bytes that did not fit in the return area and still remain in the buffer. Another READA request will retrieve that data. Certain IFI requests return a known quantity of information. Other requests will terminate when the return area is full. |
| | 1C | Four-byte signed integer | Reserved. |
| IFCARLC | 20 | Four-byte signed integer | Indicates the number of records lost prior to a READA call. Records are lost when the OP buffer storage is exhausted before the contents of the buffer are transferred to the application program via an IFI READA request. Records that do not fit in the OP buffer are not written and are counted as records lost. |
| IFCAOPN | 24 | Character, 4 bytes | Destination name used on a READA request. This field identifies the buffer requested, and is required on a READA request. Your monitor program must set this field. The instrumentation facility fills in this field on START TRACE to an OP*n* destination from an monitor program. If your monitor program started multiple OP*n* destination traces, the first one is placed in this field. If your monitor program did not start an OP*n* destination trace, the field is not modified. The OP*n* destination and owner ID are used on subsequent READA calls to find the asynchronous buffer. |
| IFCAOPNL | 28 | Two-byte signed integer | Length of the OP*n* destinations started. On any command entered by IFI, the value is set to X'0004'. If an OP*n* destination is started, the length is incremented to include all OP*n* destinations started. |
| | 2A | Two-byte signed integer | Reserved. |
| IFCAOPNR | 2C | Character, 8 fields of 4 bytes each | Space to return 8 OP*n* destination values. |
| IFCATNOL | 4C | Two-byte signed integer | Length of the trace numbers plus 4. On any command entered by IFI the value is set to X'0004'. If a trace is started, the length is incremented to include all trace numbers started. |
| | 4E | Two-byte signed integer | Reserved. |
| IFCATNOR | 50 | Character, 8 fields of 2 bytes each. | Space to hold up to eight EBCDIC trace numbers that were started. The trace number is required if the MODIFY TRACE command is used on a subsequent call. |
| IFCADL | 60 | Hex, 2 bytes | Length of diagnostic information. |
| | 62 | Hex, 2 bytes | Reserved. |

*Table 185. Instrumentation facility communication area (continued). The IFCA is mapped by assembler mapping macro DSNDIFCA.*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| IFCADD | 64 | Character, 80 bytes | Diagnostic information.<br>• IFCAFCI, offset 64, 6 bytes<br>This contains the RBA of the first CI in the active log if IFCARC2 is 00E60854. See “Reading specific log records (IFCID 0129)” on page 968 for more information.<br>• IFCAR0, offset 6C, 4 bytes<br>For COMMAND requests, this field contains -1 or the return code from the component that executed the command.<br>• IFCAR15, offset 70, 4 bytes<br>For COMMAND requests, this field contains one of the following values:<br>**0** The command completed successfully.<br>**4** Internal error.<br>**8** The command was not processed because of errors in the command.<br>**12** The component that executed the command returned the return code in IFCAR0.<br>**16** An abend occurred during command processing. Command processing might be incomplete, depending on when the error occurred. See IFCAR0 for more information.<br>**20** Response buffer storage was not available. The command completed, but no response messages are available. See IFCAR0 for more information.<br>**24** Storage was not available in the DSNMSTR address space. The command was not processed.<br>**28** CSA storage was not available. If a response buffer is available, the command might have partially completed. See IFCAR0 for more information.<br>**32** The user is not authorized to issue the command. The command was not processed.<br>• IFCAGBPN, offset 74, 8 bytes<br>This is the group buffer pool name in error if IFCARC2 is 00E60838 or 00E60860<br>• IFCABSRQ, offset 88, 4 bytes<br>This is the size of the return area required when the reason code is 00E60864.<br>• IFCAHLRS, offset 8C, 6 bytes<br>This field can contain the highest LRSN or log RBA in the active log (when WQALLMOD is 'H'). Or, it can contain the RBA of the log CI given to the Log Exit when the last CI written to the log was not full, or an RBA of zero (when WQALLMOD is 'P'). |
| IFCAGRSN | 98 | Four-byte signed integer | Reason code for the situation in which an IFI calls requests data from members of a data sharing group, and not all the data is returned from group members. See Part 3 of *DB2 Messages and Codes* for information about reason codes. |

*Table 185. Instrumentation facility communication area (continued). The IFCA is mapped by assembler mapping macro DSNDIFCA.*

| Name | Hex offset | Data type | Description |
|---|---|---|---|
| IFCAGBM | 9C | Four-byte signed integer | Total length of data that was returned from other data sharing group members and fit in the return area. |
| IFCAGBNM | A0 | Four-byte signed integer | Total length of data that was returned from other data sharing group members and did not fit in the return area.. |
| IFCADMBR | A4 | Character, 8 bytes | Name of a single data sharing group member on which an IFI request is to be executed. Otherwise, this field is blank. If this field contains a member name, DB2 ignores field IFCAGLBL. |
| IFCARMBR | AC | Character, 8 bytes | Name of the data sharing group member from which data is being returned. DB2 sets this field in each copy of the IFCA that it places in the return area, not in the IFCA of the application that makes the IFI request. |

## Return area

You must specify a return area on all READA, READS, and COMMAND requests. IFI uses the return area to return command responses, synchronous data, and asynchronous data to the monitor program.

*Table 186. Return area*

| Hex offset | Data type | Description |
|---|---|---|
| 0 | Signed four-byte integer | The length of the return area, plus 4. This must be set by the monitor program. The valid range for READA requests is 100 to 1048576 (X’00000064’ to X’00100000’). The valid range for READS requests is 100 to 2147483647 (X’00000064’ to X’7FFFFFFF’). |
| 4 | Character, varying-length | DB2 places as many varying-length records as it can fit into the area following the length field. The monitor program’s length field is not modified by DB2. Each varying-length trace record has a 2-byte length field.<br>After a COMMAND request, the last character in the return area is a new-line character (X'15'). |

*Table 187. Return area using IFCID 306*

| Hex | Data type | Description |
|---|---|---|
| 0 | Signed four-byte integer | The length of the return area |
| 4 | Character, 4 bytes | The eye-catcher, a constant, I306. Beginning of QW0306OF mapping. |
| 8 | Character, 60 bytes. | Reserved |
| 44 | Signed four-byte integer | The length of the returned data. |

**Note:** For more information about reading log records, see “Appendix C. Reading log records” on page 957

The destination header for data returned on a READA or READS request is mapped by macro DSNDQWIW or the header QW0306OF for IFCID 306 requests. Please refer to *prefix*.SDSNSAMP(DSNWMSGS) for the format of the trace record and its header. The size of the return area for READA calls should be as large as the buffer specified on the BUFSIZE keyword when the trace is started.

Data returned on a COMMAND request consists of varying-length segments (X'xxxxrrrr' where the length is 2 bytes and the next 2 bytes are reserved), followed by the message text. More than one record can be returned. The last character in the return area is a new-line character (X'15').

The monitor program must compare the number of bytes moved (IFCABM in the IFCA) to the sum of the record lengths to determine when all records have been processed.

## IFCID area

You must specify the IFCID area on READS and WRITE requests. The IFCID area contains the IFCIDs to process.

*Table 188. IFCID area*

| Hex Offset | Data type | Description |
|---|---|---|
| 0 | Signed two-byte integer | Length of the IFCID area, plus 4. The length can range from X'0006' to X'0044'. For WRITE requests, only one IFCID is allowed, so the length must be set to X'0006'.<br>For READS requests, you can specify multiple IFCIDs. If so, you must be aware that the returned records can be in a different sequence than requested and some records can be missing. |
| 2 | Signed two-byte integer | Reserved. |
| 4 | Hex, *n* fields of 2 bytes each | The IFCIDs to be processed. Each IFCID is placed contiguous to the previous IFCID for a READS request. The IFCIDs start at X'0000' and progress upward. You can use X'FFFF' to signify the last IFCID in the area to process. |

## Output area

The output area is used on command and WRITE requests. The area can contain a DB2 command or information to be written to the instrumentation facility. The first two bytes of area contain the length of the monitor program's record to write or the DB2 command to be issued, plus 4 additional bytes. The next two bytes are reserved. You can specify any length from 10 to 4096 (X'000A0000' to X'10000000'). The rest of the area is the actual command or record text.

For example, a START TRACE command is formatted as follows in an assembler program:

```
DC   X'002A0000'       LENGTH INCLUDING LL00 + COMMAND
DC   CL38'-STA TRACE(MON) DEST(OPX) BUFSIZE(32)  '
```

## Using IFI in a data sharing group

You can use IFI READA and READS calls in an application that runs on one member of a data sharing group to gather trace data from other members of the data sharing group. You can also use an IFI COMMAND call to execute a command at another member of a data sharing group. In addition to the IFCA fields that you use for an IFI request for a single subsystem, you need to set or read the following fields for an IFI request for a data sharing group:

**IFCAGLBL**
Set this flag on to indicate that the READS or READA request should be sent to all members of the data sharing group.

**IFCADMBR**
If you want an IFI READS, READA, or COMMAND request to be executed at a single member of the data sharing group, assign the name of the group member to this field. If you specify a name in this field, DB2 ignores IFCAGLBL.

Setting the IFCADMBR field and issuing an IFI COMMAND request is a useful way to issue a DB2 command that does not support SCOPE(GROUP) at another member of a data sharing group.

If the member whose name you specify is not active when DB2 executes the IFI request, DB2 returns an error.

**IFCARMBR**
The name of the data sharing member that generated the data that follows the IFCA. DB2 sets this value in the copy of the IFCA that it places in the requesting program's return area.

**IFCAGRSN**
A reason code that DB2 sets when not all data is returned from other data sharing group members. See Part 3 of *DB2 Messages and Codes* for specific reason codes.

**IFCAGBM**
The number of bytes of data that is returned from other members of the data sharing group and fits in the requesting program's return area.

**IFCAGBNM**
The number of bytes of data that is returned from other members of the data sharing group and *does not* fit in the requesting program's return area.

As with READA or READS requests for single DB2 subsystems, you need to issue a START TRACE command before you issue the READA or READS request. You can issue START TRACE with the parameter SCOPE(GROUP) to start the trace at all members of the data sharing group. For READA requests, you specify DEST(OPX) in the START TRACE command. DB2 collects data from all data sharing members and returns it to the OPX buffer for the member from which you issued the READA request.

If a new member joins a data sharing group while a trace with SCOPE(GROUP) is active, the trace starts at the new member.

After you issue a READS or READA call for all members of a data sharing group, DB2 returns data from all members in the requesting program's return area. Data from the local member is first, followed by the IFCA and data for all other members. For example, if the local DB2 is called DB2A, and the other two members in the group are DB2B and DB2C, the return area looks like this:

```
Data for DB2A
IFCA for DB2B (DB2 sets IFCARMBR to DB2B)
Data for DB2B
IFCA for DB2C (DB2 sets IFCARMBR to DB2C)
Data for DB2C
```

If an IFI application requests data from a single other member of a data sharing group (IFCADMBR contains a member name), the requesting program's return area contains the data for that member but no IFCA for the member. All information about the request is in the requesting program's IFCA.

Because a READA or READS request for a data sharing group can generate much more data than a READA or READS request for a single DB2, you need to increase the size of your return area to accommodate the additional data.

## Interpreting records returned by IFI

The following section describes the format of the records returned by IFI as a result of READA, READS, and COMMAND requests.

## Trace data record format

Trace records returned from READA and READS requests contain:

- A writer header that reports the length of the entire record, whether the record was in the first, middle, or last section of data, and other specific information for the writer.

  The writer header for IFI is mapped by DSNDQWIW or the header QW0306OF for IFCID 306 requests. See the mapping macros in *prefix*.SDSNMACS for the formats.
- A self-defining section
- A product section containing specific DB2 information based on the active trace
- Data areas containing the actual recorded data are mapped by multiple mapping macros described in *prefix*.SDSNMACS.

For detailed information about the format of trace records and their mapping macros, see “Appendix D. Interpreting DB2 trace output” on page 981, or see the mapping macros in *prefix*.SDSNMACS.

The following example, in dump format, shows the return area after a READS request successfully executed.

```
DFSERA10 - PRINT PROGRAM
   :

          A        B                  C
 000000   05A80000 00000510  00980001 00000054   00B80001 0000010C  01000001 0000020C
 000020   01160001 00000324  01B00001 000004D4   00000000 000004D4  00080001 000004DC
                                                          D
 000040   00010001 000004E0  00000000 000004E0   00300001 80000018  00000010 000003E8
 000060   00640064 000A0028  003D0000 0000A000   00033000 00033000  00010000 E0000000
 000080   00000000 00000000  00000000 C1C4D4C6   F0F0F140 F0F20080  00003084 00000000
 0000A0   00002000 0005003C  0028F040 40404040   40404040 40404040  40404040 40404040

   :

 000320   B0000000 202701D0  E2D7D9D4 D2C4C4F0   F0F1F940 01980064  00000000 E7C14000
 000340   00400280 C5E2E8E2  C1C4D440 40000000   000E1000 000001BC  000001B0 C9C2D4E4
 000360   E2C5D940 C9D9D3D4  D7D9D6C3 C9D9D3D4   0000003C 0000012C  0000000A 8080008C
 000380   00FA0000 00007D00  000A0014 00050028   000E0002 00080008  00400077 00000514
 0003A0   000003E8 012C0000  0000000E 000A01F4   00FA0000 00000032  000003E8 00002710
 0003C0   E2E8E2C1 C4D44040  E2E8E2D6 D7D94040   E2E8E2D6 D7D94040  000A0080 00140000
 0003E0   00000080 0005000A  13880078 0008000A   00040004 00040005  0001000A 00020005
 000400   00003000 00007800  00000001 000007D0   00040400 00780078  00010003 00019000
 000420   0000000A 00000020  00000019 00000000   0005000A 0006000A  00640064 00040063
 000440   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
 000460   30C4E2D5 D9C7C3D6  D3C4E2D5 6DD9C5C7   C9E2E3C5 D96DC1D7  D7D3C4E2 D56DD9C5
 000480   C7C9E2E3 C5D96DD6  C2D1E380 C4E2D5D9   C7C6C4C2 000009FD  C5000000 00001060
 0004A0   00020000 00001000  40000000 00000000   00000000 00000000  00000000 00000000
 0004C0   00000000 00000000  00000000 F1F161F1   F361F9F2 C4E2D5C3  F3F1F040 80000000
 0004E0   00160030 C6C1C340  00010000 C4C4C640   40404040 C1800002  00000000 C1C3E3C9
                                                 E        F
 000500   E5C54040 00000000  00000000 00000000   004C011A 006A0A31  00B45B78 E2E2D6D7
 000520   A6E9C7D5 EBDB1104  00000008 00000002   00000001 E2C1D5E3  C16DE3C5 D9C5E2C1
 000540   6DD3C1C2 C4C2F2D5  C5E34040 D3E4D5C4   F0404040 A6E9C7D2  E73C0001 004C0200
 000560   E2E8E2C1 C4D44040  D4D7C9E3 E2F14040   40404040 C2C1E3C3  C8404040 C4E2D5C5
 000580   C4C3D340 E2E8E2C1  C4D44040 00000001   00000000 00000000  00000000 00000000
 0005A0   00000000 00000000
```

*Figure 148. Example of IFI return area after READS request (IFCID 106). This output was assembled by a user-written routine and printed with the DFSERA10 print program of IMS.*

| Figure label | Description |
|---|---|
| A 05A8 | Length of record. The next two bytes are reserved. |
| B 00000510 | Offset to product section standard header. |
| C 00000054 | Offset to first data section. |
| D 80000018 | Beginning of first data section. |
| E 004C011A | Beginning of product section standard header. |
| F 006A | IFCID (decimal 106). |

For more information on IFCIDs and mapping macros, see “DB2 trace” on page 1033 and “Appendix D. Interpreting DB2 trace output” on page 981.

## Command record format

The record returned from a command request can contain none or many message text segments. Each segment is a varying-length message (LLZZ, where LL is the 2-byte length and ZZ is a 2-byte reserved area) followed by message text. The IFCA’s IFCABM field contains the total number of bytes moved.

The following example, in dump format, shows the return area after a START TRACE command successfully executed.

```
DFSERA10 - PRINT PROGRAM
   :

           A        B         C         D
 000000   007E0000 0000007A  003C0001 C4E2D5E6   F1F3F0C9 406F40D4  D6D540E3 D9C1C3C5
 000020   40E2E3C1 D9E3C5C4  6B40C1E2 E2C9C7D5   C5C440E3 D9C1C3C5  40D5E4D4 C2C5D940
                    E         F
 000040   F0F24015 003A0001  C4E2D5F9 F0F2F2C9   406F40C4 E2D5E6E5  C3D4F140 7D60E2E3
 000060   C1D9E340 E3D9C1C3  C57D40D5 D6D9D4C1   D340C3D6 D4D7D3C5  E3C9D6D5 4015
```

*Figure 149. Example of IFI return area after a START TRACE command. This output was assembled with a user-written routine and printed with DFSERA10 program of IMS.*

| Figure label | Description |
|---|---|
| A 007E0000 | Field entered by print program |
| B 0000007A | Length of return area |
| C 003C | Length of record (003C). The next two bytes are reserved. |
| D C4E2D5E6 | Beginning of first message |
| E 003A | Length of record. The next two bytes are reserved. |
| F C4E2D5F9 | Beginning of second message |

The IFCABM field in the IFCA would indicate that X'00000076' ( C + E ) bytes have been moved to the return area.

## Data integrity

Although IFI displays DB2 statistics, agent status, and resource status data, it does not change or display DB2 database data. When a process retrieves data, information is moved from DB2 fetch-protected storage to the user's address space, or from the address space to DB2 storage, in the storage key of the requester. Data moved by the READA request is serialized so that only *clean data* is moved to the address space of the requester.

The serialization techniques used to obtain data for a given READA request could have a minor performance impact on processes that are storing data into the instrumentation facility buffer simultaneously. Failures during the serialization process are handled by DB2.

The DB2 structures searched on a READS request are validated before they are used. If the DB2 structures are updated while being searched, inconsistent data might be returned. If the structures are deleted while being searched, users might access invalid storage areas, causing an abend. If an abend does occur, the functional recovery routine of the instrumentation facility traps the abend and returns information about it to the application program's IFCA.

## Auditing data

Starting, stopping, or modifying trace through IFI might cause changes to the events being traced for audit. Each time these trace commands are processed a record is sent to the destination processing the trace type. In the case of audit, the audit destination receives a record indicating a trace status has been changed. These records are IFCID 0004 and 0005.

## Locking considerations

When designing your application to use IFI, you need to consider the potential for locking delays, deadlocks, and time-out conflicts. Locks are obtained for IFI in the following situations:

- When READS and READA requests are checked for authorization, short duration locks on the DB2 catalog are obtained. When the check is made, subsequent READS or READA requests are not checked for authorization. Remember, if you are using the access control exit routine, then that routine might be controlling the privileges that the monitor trace can use.
- When DB2 commands are submitted, each command is checked for authorization. DB2 database commands obtain additional locks on DB2 objects.

A program can issue SQL statements through an attachment facility and DB2 commands through IFI. This environment creates the potential for an application to deadlock or time-out with itself over DB2 locks acquired during the execution of SQL statements and DB2 database commands. You should ensure that all DB2 locks acquired by preceding SQL statements are no longer held when the DB2 database command is issued. You can do this by:

- Binding the DB2 plan with ACQUIRE(USE) and RELEASE(COMMIT) bind parameters
- Initiating a commit or rollback to free any locks your application is holding, before issuing the DB2 command

If you use SQL in your application, the time between commit operations should be short. For more information on locking, see “Chapter 30. Improving concurrency” on page 643.

## Recovery considerations

When an application program issues an IFI call, the function requested is immediately performed. If the application program subsequently abends, the IFI request is *not* backed out. In contrast, requests that do not use IFI are committed and abended as usual. For example, if an IFI application program also issues SQL calls, a program abend causes the SQL activity to be backed out.

## Errors

While using IFI, you might encounter any of these types of error:

- Connection failure, because the user is not authorized to connect to DB2
- Authorization failure, because the process is not authorized to access the DB2 resources specified

Requests sent through IFI can fail for a variety of reasons, including:

- One or more parameters are invalid.
- The IFCA area is invalid.
- The specified OP*n* is in error.
- The requested information is not available.
- The return area is too small.

Return code and reason code information is stored in the IFCA in fields IFCARC1 and IFCARC2. Further return and reason code information is contained in Part 3 of *DB2 Messages and Codes*.

# Appendix F. Using tools to monitor performance

This section describes the various facilities for monitoring DB2 activity and performance. It includes information on facilities within the DB2 product as well as tools available outside of DB2. Figure 150 shows various monitoring tools that can be used in a DB2 environment.

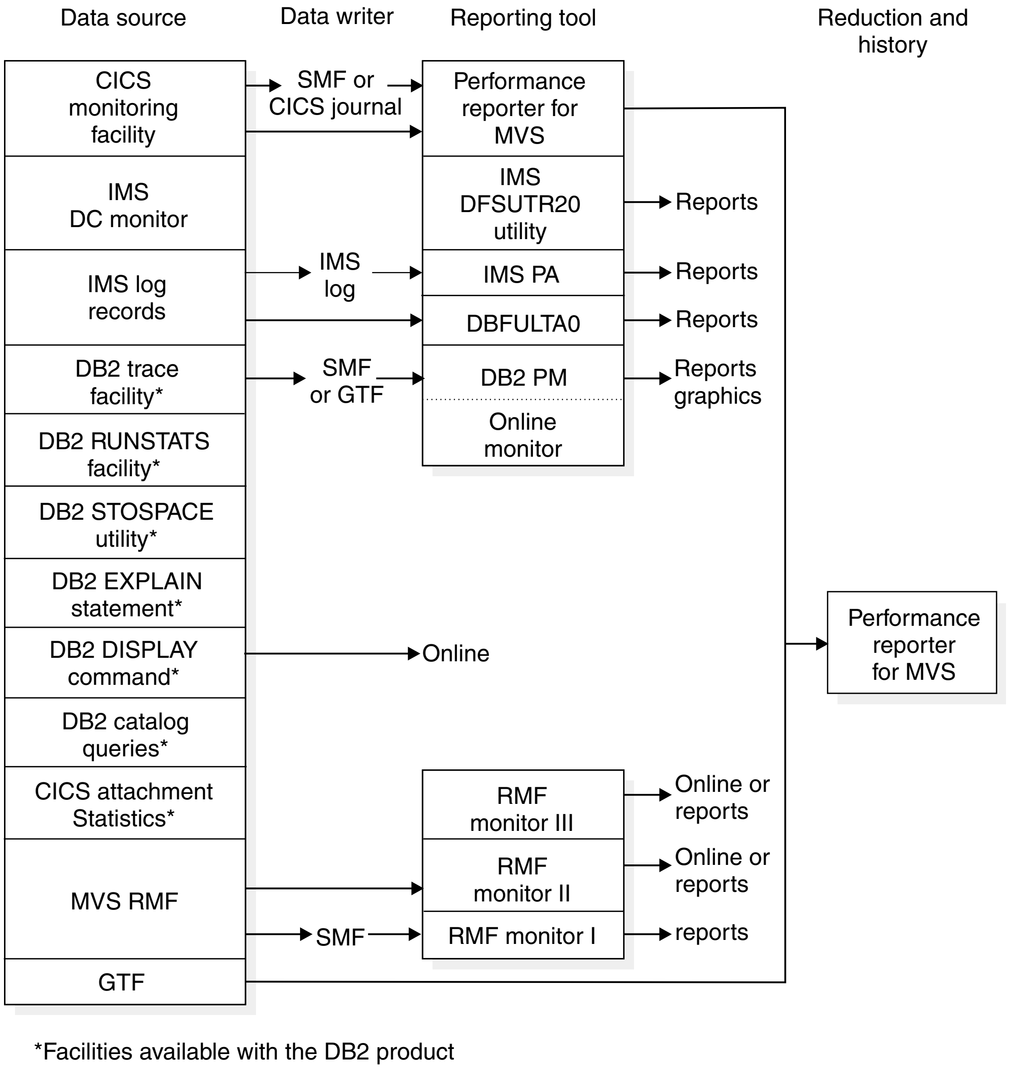


*Figure 150. Monitoring tools in a DB2 environment*

**CICS Monitoring Facility (CMF)** provides performance information about each CICS transaction executed. It can be used to investigate the resources used and the time spent processing transactions. Be aware that overhead is significant when CMF is used to gather performance information.

| **IMS Performance Analyzer (IMS PA)**, a separately licensed program, can be used to produce transit time information based on the IMS log data set. It can also be used to investigate response-time problems of IMS DB2 transactions.

**Fast Path Log Analysis Utility (DBFULTA0)**, an IMS utility, provides performance data.

**DB2 trace facility** provides DB2 performance and accounting information. It is described under “DB2 trace” on page 1033.

**System Management Facility (SMF)** is an MVS service aid used to collect information from various MVS subsystems. This information is dumped and reported periodically, such as once a day. Refer to “Recording SMF trace data” on page 1037 for more information.

**Generalized Trace Facility (GTF)** is an MVS service aid that collects information to analyze particular situations. GTF can also be used to analyze seek times and Supervisor Call instruction (SVC) usage, and for other services. See “Recording GTF trace data” on page 1039 for more information.

**DB2 Performance Monitor (DB2 PM)** is an orderable feature of DB2 used to analyze DB2 trace records. DB2 PM is described under “DB2 Performance Monitor (DB2 PM)” on page 1039.

**DB2 RUNSTATS utility** can report space use and access path statistics in the DB2 catalog. See “Gathering monitor and update statistics” on page 775 and Part 2 of *DB2 Utility Guide and Reference*.

**DB2 STOSPACE utility** provides information about the actual space allocated for storage groups, table spaces, table space partitions, index spaces, and index space partitions. See in Part 2 of *DB2 Utility Guide and Reference*.

**DB2 EXPLAIN statement** provides information about the access paths used by DB2. See “Chapter 33. Using EXPLAIN to improve SQL performance” on page 789 and Chapter 5 of *DB2 SQL Reference*.

**DB2 DISPLAY command** gives you information about the status of threads, databases, buffer pools, traces, allied subsystems, applications, and the allocation of tape units for the archive read process. For information about the DISPLAY BUFFERPOOL command, see “Monitoring and tuning buffer pools using online commands” on page 563. For information about using the DISPLAY command to monitor distributed data activity, see “Using the DISPLAY command” on page 866. For the detailed syntax of each command, refer to Chapter 2 of *DB2 Command Reference*.

| **DB2 Connect** can monitor and report DB2 server-elapsed time for client
| applications that access DB2 data. See “Reporting server-elapsed time” on
| page 870.

**Performance Reporter for MVS**, formerly known as EPDM, is a licensed program that collects SMF data into a DB2 database and allows you to create reports on the data. See “Performance Reporter for MVS” on page 1040.

**DB2 catalog queries** help you determine when to reorganize table spaces and indexes. See the description of the REORG utility in Part 2 of *DB2 Utility Guide and Reference*.

**CICS Attachment Facility statistics** provide information about the use of CICS threads. This information can be displayed on a terminal or printed in a report.

**Resource Measurement Facility (RMF)** is an optional feature of OS/390 that provides system-wide information on processor utilization, I/O activity, storage, and paging. There are three basic types of RMF sessions: Monitor I, Monitor II, and Monitor III. Monitor I and Monitor II sessions collect and report data primarily about specific system activities. Monitor III sessions collect and report data about overall system activity in terms of work flow and delay.

## Using MVS, CICS, and IMS tools

To monitor DB2 and CICS, you can use:
- RMF Monitor II for physical resource utilizations
- GTF for detailed I/O monitoring when needed

- Performance Reporter for MVS for application processor utilization, transaction performance, and system statistics.

You can use RMF Monitor II to dynamically monitor system-wide physical resource utilizations, which can show queuing delays in the I/O subsystem.

In addition, the CICS attachment facility DSNC DISPLAY command allows any authorized CICS user to dynamically display statistical information related to thread usage and situations when all threads are busy. For more information about the DSNC DISPLAY command, see Chapter 2 of *DB2 Command Reference*.

Be sure that the number of threads reserved for specific transactions or for the pool is large enough to handle the actual load. You can dynamically modify the value specified in the resource control table (RCT) with the DSNC MODIFY TRANSACTION command. You might also need to modify the maximum number of threads specified for the MAX USERS field on installation panel DSNTIPE.

To monitor DB2 and IMS, you can use:

- RMF Monitor II for physical resource utilizations
- GTF for detailed I/O monitoring when needed
- IMS Performance Analyzer, or its equivalent, for response-time analysis and tracking all IMS-generated requests to DB2
- Fast Path Log Analysis Utility (DBFULTA0) for performance data

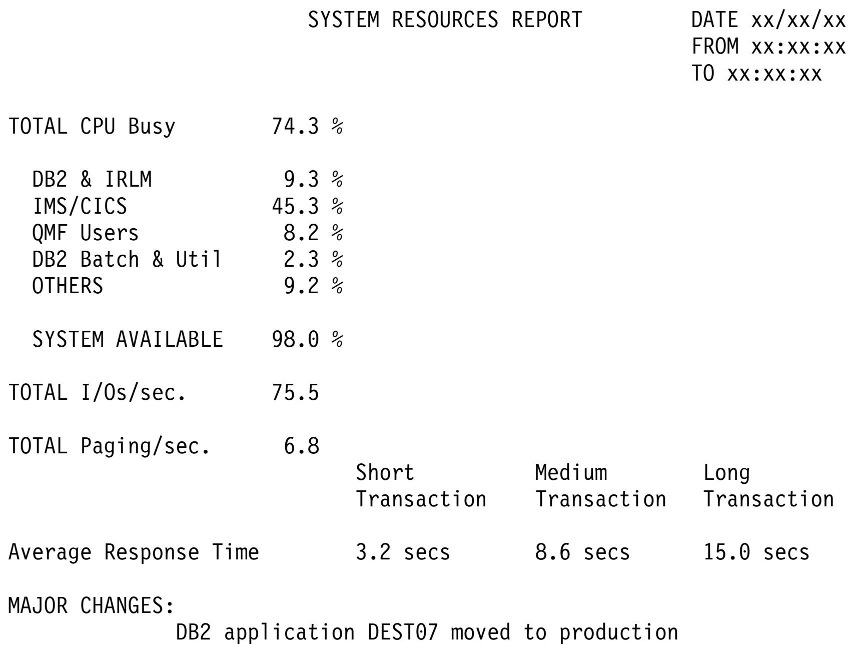
In addition, the DB2 IMS attachment facility allows you to use the DB2 command DISPLAY THREAD command to dynamically observe DB2 performance.

## Monitoring system resources

Monitor system resources to:

- Detect resource constraints (processor, I/O, storage)
- Determine how resources are consumed
- Check processor, I/O, and paging rate to detect a bottleneck in the system
- Detect changes in resource use over comparable periods.

Figure 151 shows an example of a suggested system resources report.



```
                          SYSTEM RESOURCES REPORT         DATE xx/xx/xx
                                                          FROM xx:xx:xx
                                                          TO xx:xx:xx

TOTAL CPU Busy        74.3 %

  DB2 & IRLM           9.3 %
  IMS/CICS            45.3 %
  QMF Users            8.2 %
  DB2 Batch & Util     2.3 %
  OTHERS               9.2 %

  SYSTEM AVAILABLE    98.0 %

TOTAL I/Os/sec.       75.5

TOTAL Paging/sec.      6.8
                                 Short          Medium        Long
                                 Transaction    Transaction   Transaction

Average Response Time            3.2 secs       8.6 secs      15.0 secs

MAJOR CHANGES:
              DB2 application DEST07 moved to production
```

*Figure 151. User-Created system resources report*

The RMF reports used to produce the information in Figure 151 were:

- The RMF CPU activity report, which lists TOTAL CPU Busy and the TOTAL I/Os per second.
- RMF paging activity report, which lists the TOTAL Paging rate per second for main storage.
- The RMF work load activity report, which is used to estimate where resources are spent. Each address space or group of address spaces to be reported on separately must have different SRM reporting or performance groups. The following SRM reporting groups are considered:
  - DB2 address spaces:
    DB2 Database Address Space (*ssnm*DBM1)
    DB2 System Services Address Space (*ssnm*MSTR)
    Distributed Data Facility (*ssnm*DIST)
    IRLM (IRLMPROC)
  - IMS or CICS
  - TSO-QMF
  - DB2 batch and utility jobs

  The CPU for each group is obtained using the ratio (A/B) × C, where:
  A is the sum of CPU and service request block (SRB) service units for the specific group
  B is the sum of CPU and SRB service units for all the groups
  C is the total processor utilization.

  The CPU and SRB service units must have the same coefficient.

  You can use a similar approach for an I/O rate distribution.

MAJOR CHANGES shows the important environment changes, such as:

- DB2 or any related software-level change
- DB2 changes in the load module for system parameters

- New applications put into production
- Increase in the number of QMF users
- Increase in batch and utility jobs
- Hardware changes

MAJOR CHANGES is also useful for discovering the reason behind different monitoring results.

## Monitoring transaction manager throughput

Use IMS or CICS monitoring facilities to determine throughput, in terms of transactions processed, and transaction response times. Depending on the transaction manager, you can use the following reports:

- IMS Performance Analyzer
- Fast Path Log Analysis Utility (DBFULTA0)
- Performance Reporter for MVS

In these reports:

- The transactions processed include DB2 and non-DB2 transactions.
- The transaction processor time includes the DB2 processor time for IMS but not for CICS.
- The transaction transit response time includes the DB2 transit time.

A historical database is useful for saving monitoring data from different periods. Such data can help you track the evolution of your system. You can use Performance Reporter for MVS or write your own application based on DB2 and QMF when creating this database.

## DB2 trace

The information under this heading, up to “Recording SMF trace data” on page 1037, is General-use Programming Interface and Associated Guidance Information as defined in “Notices” on page 1095.

DB2’s instrumentation facility component (IFC) provides a trace facility that you can use to record DB2 data and events. With the IFC, however, analysis and reporting of the trace records must take place outside of DB2. You can use the IBM DATABASE 2 Performance Monitor (DB2 PM) feature of DB2, to format, print, and interpret DB2 trace output. You can view an online snapshot from trace records by using DB2 PM or other online monitors. For more information on DB2 PM, see *DB2 PM for OS/390 General Information*. For the exact syntax of the trace commands see Chapter 2 of *DB2 Command Reference*.

If you do not have DB2 PM, or if you want to do your own analysis of the DB2 trace output, refer to “Appendix D. Interpreting DB2 trace output” on page 981. Also consider writing your own program using the instrumentation facility interface (IFI). Refer to “Appendix E. Programming for the Instrumentation Facility Interface (IFI)” on page 997 for more information on using IFI.

Each *trace class* captures information on several subsystem events. These events are identified by many instrumentation facility component identifiers (IFCIDs). The IFCIDs are described by the comments in their mapping macros, contained in *prefix*.SDSNMACS, which is shipped to you with DB2.

# Types of traces

DB2 trace can record six types of data: statistics, accounting, audit, performance, monitor, and global. The description of the START TRACE command in Chapter 2 of *DB2 Command Reference* indicates which IFCIDs are activated for the different types of trace and the classes within those trace types. For details on what information each IFCID returns, see the mapping macros in *prefix*.SDSNMACS.

The trace records are written using GTF or SMF records. See “Recording SMF trace data” on page 1037 and “Recording GTF trace data” on page 1039 before starting any traces. Trace records can also be written to storage, if you are using the monitor trace class.

## Statistics trace

The statistics trace reports information about how much the DB2 system services and database services are used. It is a system-wide trace and should not be used for chargeback accounting. Use the information the statistics trace provides to plan DB2 capacity, or to tune the entire set of active DB2 programs.

Statistics trace classes 1, 3, 4, and 5 are the default classes for the statistics trace if statistics is specified YES in panel DSNTIPN. If the statistics trace is started using the START TRACE command, then class 1 is the default class.

- Class 1 provides information about system services and database statistics. It also includes the system parameters that were in effect when the trace was started.
- Class 3 provides information about deadlocks and timeouts.
- Class 4 provides information about exceptional conditions.
- Class 5 provides information about data sharing.

If you specified YES in the SMF STATISTICS field on the Tracing Panel (DSNTIPN), the statistics trace starts automatically when you start DB2, sending class 1, 3, 4 and 5 statistics data to SMF. SMF records statistics data in both SMF type 100 and 102 records. IFCIDs 0001, 0002, 0202, and 0230 are of SMF type 100. All other IFCIDs in statistics trace classes are of SMF type 102. From panel DSNTIPN, you can also control the statistics collection interval (STATISTICS TIME field).

The statistics trace is written on an interval basis, and you can control the exact time that statistics traces are taken.

## Accounting trace

The DB2 accounting trace provides information related to application programs, including such things as:

Start and stop times<br>
Number of commits and aborts<br>
The number of times certain SQL statements are issued<br>
Number of buffer pool requests<br>
Counts of certain locking events<br>
Processor resources consumed<br>
Thread wait times for various events<br>
RID pool processing<br>
Distributed processing<br>
Resource limit facility statistics

DB2 trace begins collecting this data at successful thread allocation to DB2, and writes a completed record when the thread terminates or when the authorization ID changes.

During CICS thread reuse, a change in the authid or transaction code initiates the sign-on process, which terminates the accounting interval and creates the accounting record. TXIDSO=NO eliminates the sign-on process when only the transaction code changes. When a thread is reused without initiating sign-on, several transactions are accumulated into the same accounting record, which can make it very difficult to analyze a specific transaction occurrence and correlate DB2 accounting with CICS accounting. However, applications that use TOKENE=YES or TOKENI=YES initiate a “partial sign-on”, which creates an accounting record for each transaction. You can use this data to perform program-related tuning and assess and charge DB2 costs.

Accounting data for class 1 (the default) is accumulated by several DB2 components during normal execution. This data is then collected at the end of the accounting period; it does not involve as much overhead as individual event tracing.

On the other hand, when you start class 2, 3, 7, or 8, many additional trace points are activated. Every occurrence of these events is traced internally by DB2 trace, but these traces are not written to any external destination. Rather, the accounting facility uses these traces to compute the additional total statistics that appear in the accounting record, IFCID 003, when class 2 or class 3 is activated. Accounting class 1 must be active to externalize the information.

To turn on accounting for packages and DBRMs, accounting trace classes 1 and 7 must be active. Though you can turn on class 7 while a plan is being executed, accounting trace information is only gathered for packages or DBRMs executed after class 7 is activated. Activate accounting trace class 8 with class 1 to collect information about the amount of time an agent was suspended in DB2 for each executed package. If accounting trace classes 2 and 3 are activated, there is minimal additional performance cost for activating accounting trace classes 7 and 8.

If you want information from either, or both, accounting class 2 and 3, be sure to activate classes 2 and/or 3 before your application starts. If these classes are activated during the application, the times gathered by DB2 trace are only from the time the class was activated.

Accounting trace class 5 provides information on the amount of elapsed time and TCB time that an agent spent in DB2 processing instrumentation facility interface (IFI) requests. If an agent did not issue any IFI requests, these fields are not included in the accounting record.

If you specified YES for SMF ACCOUNTING on the Tracing Panel (DSNTIPN), the accounting trace starts automatically when you start DB2, and sends IFCIDs that are of SMF type 100 to SMF. The accounting record IFCID 0003 is of SMF type 101.

## Audit trace

The audit trace collects information about DB2 security controls and is used to ensure that data access is allowed only for authorized purposes. On the CREATE TABLE or ALTER TABLE statements, you can specify whether or not a table is to be audited, and in what manner; you can also audit security information such as any access denials, grants, or revokes for the table. The default causes no auditing to take place. For descriptions of the available audit classes and the events they trace, see “Audit class descriptions” on page 220.

If you specified YES for AUDIT TRACE on the Tracing Panel (DSNTIPN), audit trace class 1 starts automatically when you start DB2. By default, DB2 will send audit data to SMF. SMF records audit data in type 102 records. When you invoke the -START TRACE command, you can also specify GTF as a destination for audit data. “Chapter 14. Auditing” on page 219 describes the audit trace in detail.

## Performance trace

The performance trace provides information about a variety of DB2 events, including events related to distributed data processing. You can use this information to further identify a suspected problem, or to tune DB2 programs and resources for individual users or for DB2 as a whole.

You cannot automatically start collecting performance data when you install or migrate DB2. To trace performance data, you must use the -START TRACE(PERFM) command. For more information about the -START TRACE(PERFM) command, refer to Chapter 2 of *DB2 Command Reference*.

The performance trace defaults to GTF.

## Monitor trace

The monitor trace records data for online monitoring with user-written programs. This trace type has several predefined classes; those that are used explicitly for monitoring are listed here:

- Class 1 (the default) allows any application program to issue an instrumentation facility interface (IFI) READS request to the IFI facility. If monitor class 1 is inactive, a READS request is denied. Activating class 1 has a minimal impact on performance.
- Class 2 collects processor and elapsed time information. The information can be obtained by issuing a READS request for IFCID 0147 or 0148. In addition, monitor trace class 2 information is available in the accounting record, IFCID 0003. Monitor class 2 is equivalent to accounting class 2 and results in equivalent overhead. Monitor class 2 times appear in IFCIDs 0147, 0148, and 0003 if either monitor trace class 2 or accounting class 2 is active.
- Class 3 activates DB2 wait timing and saves information about the resource causing the wait. The information can be obtained by issuing a READS request for IFCID 0147 or 0148. In addition, monitor trace class 3 information is available in the accounting record, IFCID 0003. As with monitor class 2, monitor class 3 overhead is equivalent to accounting class 3 overhead.

  When monitor trace class 3 is active, DB2 can calculate the duration of a class 3 event, such as when an agent is suspended due to an unavailable lock. Monitor class 3 times appear in IFCIDs 0147, 0148, and 0003, if either monitor class 3 or accounting class 3 is active.
- Class 5 traces the amount of time spent processing IFI requests.
- Class 7 traces the amount of time an agent spent in DB2 to process each package. If monitor trace class 2 is active, activating class 7 has minimal performance impact.
- Class 8 traces the amount of time an agent was suspended in DB2 for each package executed. If monitor trace class 3 is active, activating class 8 has minimal performance impact.

For more information on the monitor trace, refer to “Appendix E. Programming for the Instrumentation Facility Interface (IFI)” on page 997.

## Effect on DB2 performance

The volume of data DB2 trace collects can be quite large. Consequently, the number of trace records you request will affect system performance. In particular, when you activate a performance trace, you should qualify the -START TRACE command with the particular classes, plans, authorization IDs, and IFCIDs you want to trace.

The following recommendations apply:

- When starting a performance trace, be sure that you know what you want to report, for example, I/O only or SQL only. See DB2 PM for examples of which classes produce which reports. Otherwise, you might have incomplete reports and have to rerun or collect too much data, overloading the data collector.
- When the statistics trace is active, statistics are collected by SMF at all times. Use the default statistics frequency of 30 minutes.
- Decide if the continuous collection of accounting data is needed. If a transaction manager provides enough accounting information, DB2 accounting might not be needed. In environments where the processor is heavily loaded, consider not running accounting on a continuous basis.
- When using accounting on a continuous basis, start classes 1 and 3 to SMF (SMF ACCOUNTING on panel DSNTIPN). You might also want to start accounting class 2 because it provides additional information that can be useful in resolving problems. Accounting class 2 does introduce some additional processor cost.
- Use the performance trace for short periods of time (START/STOP TRACE) and restrict it to certain users, applications, and classes. Use the default destination GTF to allow immediate analysis of the trace information.
- Start the global trace only if a problem is under investigation, and IBM service personnel have requested a trace.

For more detailed information about the amount of processor resources consumed by DB2 trace, see “Reducing the amount of processor resources consumed” on page 544.

## Recording SMF trace data

Each location is responsible for processing the SMF records produced by DB2 trace.

For example, during DB2 execution, you can use the MVS operator command SETSMF or SS to alter SMF parameters you specified previously. The following command records statistics (record type 100), accounting (record type 101), and performance (record type 102) data to SMF. To execute this command, specify PROMPT(ALL) or PROMPT(LIST) in the SMFPRM*xx* member used from SYS1.PARMLIB.

```
SETSMF SYS(TYPE(100:102))
```

If you are not using measured usage licensing, do not specify type 89 records or you will incur the overhead of collecting that data.

You can use the SMF program IFASMFDP to dump these records to a sequential data set. You might want to develop an application or use DB2 PM to process these records. For a sample DB2 trace record sent to SMF, see Figure 142 on page 983. For more information about SMF, refer to *OS/390 JES2 Initialization and Tuning Guide*.

## Activating SMF

SMF must be running before you can send data to it. To make it operational, update member SMFPRM*xx* of SYS1.PARMLIB, which indicates whether SMF is active and which types of records SMF accepts. For member SMFPRM*xx*, *xx* are two user-defined alphanumeric characters appended to 'SMFPRM' to form the name of an SMFPRM*xx* member. To update this member, specify the ACTIVE parameter and the proper TYPE subparameter for SYS and SUBSYS.

You can also code an IEFU84 SMF exit to process the records that are produced.

## Allocating additional SMF buffers

When you specify a performance trace type, the volume of data that DB2 can collect can be quite large. If you are sending this data to SMF, you must allocate adequate SMF buffers; the default buffer settings will probably be insufficient.

If an SMF buffer shortage occurs, SMF rejects any trace records sent to it. DB2 sends a message (DSNW133I) to the MVS operator when this occurs. DB2 treats the error as temporary and remains active even though data could be lost. DB2 sends another message (DSNW123I) to the MVS operator when the shortage has been alleviated and trace recording has resumed.

You can determine if trace data has been lost by examining the DB2 statistics records with an IFCID of 0001, as mapped by macro DSNQWST. These records show:

- The number of trace records successfully written
- The number of trace records that could not be written
- The reason for the failure

If your location uses SMF for performance data or global trace data, be sure that:

- Your SMF data sets are large enough to hold the data.
- SMF is set up to accept record type 102. (Specify member SMFPRM*xx*, for which 'xx' are two user-defined alphanumeric characters.)
- Your SMF buffers are large enough.

Specify SMF buffering on the VSAM BUFSP parameter of the access method services DEFINE CLUSTER statement. Do not use the default settings if DB2 performance or global trace data is sent to SMF. Specify CISZ(4096) and BUFSP(81920) on the DEFINE CLUSTER statement for each SMF VSAM data set. These values are the minimum required for DB2; you might have to increase them, depending on your MVS environment.

DB2 runs above the 16MB line of virtual storage in a cross-memory environment.

## Reporting data in SMF

There are several ways to report trace records sent to SMF:

- Use Performance Reporter for MVS to collect the data and create graphical or tabular reports.
- Write an application program to read and report information from the SMF data set. You can tailor it to fit your exact needs.
- Use DB2 PM. See “DB2 Performance Monitor (DB2 PM)” on page 1039 for a discussion of DB2 PM’s capabilities.

In any of those ways you can compare any report for a current day, week, or month with an equivalent sample, as far back as you want to go. The samples become more widely spaced but are still available for analysis.

## Recording GTF trace data

The default destination for the performance trace classes is the generalized trace facility (GTF). The MVS operator must start GTF before you can send data to it. When starting GTF, specify TIME=YES, and then TRACE=USRP. Start GTF as follows to ensure that offsets map correctly. Be sure that no GTF member exists in SYS1.PARMLIB.

| You enter... | System responds... |
|---|---|
| S GTF,,,(TIME=YES) | AHL100A SPECIFY TRACE OPTIONS |
| TRACE=USRP | AHL101A SPECIFY TRACE EVENT KEYWORDS --USR= |
| USR=(FB9) | AHL102A CONTINUE TRACE DEFINITION OR REPLY END |
| END | AHL125A RESPECIFY TRACE OPTIONS OR REPLY U |
| U | AHL031I GTF INITIALIZATION COMPLETE |

**Note:** To make stopping GTF easier, you can give the GTF session a name when you start it. For example, you could specify `S GTF.GTF,,,(TIME=YES)`.

If a GTF member exists in SYS1.PARMLIB, the GTF trace option USR might not be in effect. When no other member exists in SYS1.PARMLIB, you are sure to have only the USR option activated, and no other options that might add unwanted data to the GTF trace.

When starting GTF, if you use the JOBNAMEP option to obtain only those trace records written for a specific job, trace records written for other agents are not written to the GTF data set. This means that a trace record that is written by a system agent that is processing for an allied agent is discarded if the JOBNAMEP option is used. For example, after a DB2 system agent performs an IDENTIFY request for an allied agent, an IFCID record is written. If the JOBNAMEP keyword is used to collect trace data for a specific job, however, the record for the IDENTIFY request is not written to GTF, even if the IDENTIFY request was performed for the job named on the JOBNAMEP keyword.

You can record DB2 trace data in GTF using a GTF event ID of X'FB9'.

Trace records longer than the GTF limit of 256 bytes are spanned by DB2. For instructions on how to process GTF records, refer to “Appendix D. Interpreting DB2 trace output” on page 981.

## DB2 Performance Monitor (DB2 PM)

DB2 PM is a performance analysis tool for DB2. Its primary objective is to report DB2 instrumentation data in a form that is easy to understand and analyze.

DB2 PM presents this instrumentation data in the following ways:

- The Batch report sets present the data you select in comprehensive reports or graphs containing system-wide and application-related information for both single DB2 subsystems and DB2 members of a data sharing group. You can combine instrumentation data from several different DB2 locations into one report.

Batch reports can be used to examine performance problems and trends over a period of time.

- The Online Monitor gives a current “snapshot” view of a running DB2 subsystem, including applications that are running. Its history function displays information about subsystem and application activity in the recent past.

See *DB2 PM for OS/390 General Information* for more information about the latest features in DB2 PM.

## Performance Reporter for MVS

Performance Reporter for MVS, formerly known as EPDM, collects data into a DB2 database and allows you to create graphical and tabular reports to use in managing systems performance. The data can come from different sources, including SMF, the IMS log, the CICS journal, RMF, and DB2.

When considering the use of Performance Reporter for MVS, consider the following:

- Performance Reporter data collection and reporting are based on user specifications. Therefore, an experienced user can produce more suitable reports than the predefined reports produced by other tools.
- Performance Reporter provides historical performance data that you can use to compare a current situation with previous data.
- Performance Reporter can be used very effectively for reports based on the DB2 statistics and accounting records. When using it for the performance trace consider that:
  - Because of the large number of different DB2 performance records, a substantial effort is required to define their formats to Performance Reporter. Changes in the records require review of the definitions.
  - Performance Reporter not handle information from paired records, such as “start event” and “end event.” These record pairs are used by DB2 PM to calculate elapsed times, such as the elapsed time of I/Os and lock suspensions.

The general recommendation for Performance Reporter and DB2 PM use in a DB2 subsystem is:

- If Performance Reporter is already used or there is a plan to use it at the location:
  - Extend Performance Reporter usage to the DB2 accounting and statistics records.
  - Use DB2 PM for the DB2 performance trace.
- If Performance Reporter is not used and there is no plan to use it:
  - Use DB2 PM for the statistics, accounting, and performance trace.
  - Consider extending DB2 PM with user applications based on DB2 and QMF, to provide historical performance data.

## Monitoring application plans and packages

The following statements identify plans and packages that:

- Possibly redo validity checks at run time; if an invalid object or missing authority is found, DB2 issues a warning and checks again for the object or authorization at run time.
- Use repeatable read.

- Are invalid (must be rebound before use), for example, the deleting an index or revoking authority can render a plan or package invalid.
- Are inoperative (require an explicit BIND or REBIND before use). A plan or package can be marked inoperative after an unsuccessful REBIND.

**General-use Programming Interface**

```
SELECT NAME, VALIDATE, ISOLATION, VALID, OPERATIVE
  FROM SYSIBM.SYSPLAN
  WHERE VALIDATE = 'R' OR ISOLATION = 'R'
    OR VALID = 'N' OR OPERATIVE = 'N';

SELECT COLLID, NAME, VERSION, VALIDATE, ISOLATION, VALID, OPERATIVE
  FROM SYSIBM.SYSPACKAGE
  WHERE VALIDATE = 'R' OR ISOLATION = 'R'
    OR VALID = 'N' OR OPERATIVE = 'N';
```

**End of General-use Programming Interface**

# Appendix G. Real-time statistics tables

The information under this heading is Product-sensitive Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

DB2 collects statistics that you can use to determine when you need to perform certain maintenance functions on your table spaces and index spaces.

DB2 collects the statistics in real time. You create tables into which DB2 periodically writes the statistics. You can then write applications that query the statistics and help you decide when to run REORG, RUNSTATS, or COPY, or enlarge your data sets. For information on a DB2-supplied stored procedure that queries the real-time statistics tables, see “The DB2 real-time statistics stored procedure” on page 1069.

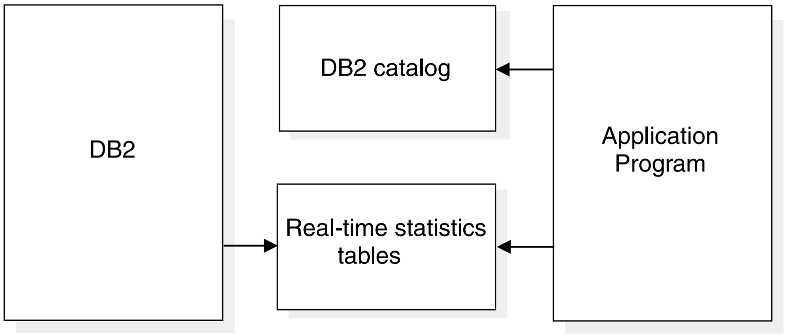


*Figure 152. Real-Time Statistics overview*

The following sections provide detailed information about the real-time statistics tables:
- “Setting up your system for real-time statistics”
- “Contents of the real-time statistics tables” on page 1045
- “Operating with real-time statistics” on page 1057

## Setting up your system for real-time statistics

DB2 always generates in-memory statistics for each table space and index space in your system. For partitioned spaces, DB2 generates information for each partition. However, you need to perform the following steps before DB2 externalizes the statistics to DB2 tables:

1. Create the real-time statistics objects. See “Creating and altering the real-time statistics objects”.
2. Set the interval for writing statistics. See “Setting the interval for writing real-time statistics” on page 1044.
3. Start the real-time statistics database. See “Starting the real-time statistics database” on page 1045.

### Creating and altering the real-time statistics objects

You need to create a database, table space, tables and indexes for the real-time statistics. Those objects are listed in Table 189 on page 1044. Use the SQL statements in member DSNTESS of data set DSN710.SDSNSAMP as a model for creating the real-time statistics objects. You can create these objects in user-managed or DB2-managed data sets.

*Restrictions on changing the provided definitions for the real-time statistics objects:* You can change most of the attributes in the provided definitions of the real-time statistics objects. However, you cannot change the following items:

- Object names

  You must use the names that are specified in DSNTESS for the database, table space, tables, indexes, and table columns.
- The CCSID parameter on the CREATE DATABASE, CREATE TABLESPACE, and CREATE TABLE statements

  The CCSID must be EBCDIC.
- Number of columns or column definitions

  You cannot add table columns or modify column definitions.

Before you can alter an object in the real-time statistics database, you must stop the database. Otherwise, you receive an SQL error.

*Table 189. DB2 objects for storing real-time statistics*

| Object name | Description |
|---|---|
| DSNRTSDB | Database for real-time statistics objects |
| DSNRTSTS | Table space for real-time statistics objects |
| SYSIBM.TABLESPACESTATS | Table for statistics on table spaces and table space partitions |
| SYSIBM.INDEXSPACESTATS | Table for statistics on index spaces and index space partitions |
| SYSIBM.TABLESPACESTATS_IX | Unique Index on SYSIBM.TABLESPACESTATS (columns DBID, PSID, and PARTITION) |
| SYSIBM.INDEXSPACESTATS_IX | Unique Index on SYSIBM.INDEXSPACESTATS (columns DBID, PSID, and PARTITION) |

To create the real-time statistics objects, you need the authority to create tables and indexes on behalf of the SYSIBM authorization ID.

DB2 inserts one row in the table for each partition or non-partitioned table space or index space. You therefore need to calculate the amount of disk space that you need for the real-time statistics tables based on the current number of table spaces and indexes in your subsystem.

To determine the amount of storage that you need for the real-time statistics when they are in memory, estimate the peak number of objects that might be updated concurrently, and multiply that total by the amount of in-memory space that DB2 uses for each object (152 bytes):

```
Amount of Storage in bytes = Maximum concurrent objects updated * 152 bytes
```

*Recommendation:* Place the statistics indexes and tables in their own buffer pool. When the statistics pages are in memory, the speed at which in-memory statistics are written to the tables improves.

## Setting the interval for writing real-time statistics

You can set the interval for writing real-time statistics when you install DB2 and update that interval online. The installation field is REAL TIME STATS on panel DSNTIPO. The default interval is 30 minutes. To update the interval, modify system parameter STATSINT.

In a data sharing environment, each member has its own interval for writing real-time statistics.

## Starting the real-time statistics database

After you create the real-time statistics database, DB2 puts it into a stopped state. After you create all the objects in the database, you need to issue START DATABASE(DSNRTSDB) to explicitly start the database.

You must start the database in read-write mode to make it possible for DB2 to externalize real-time statistics. See “When DB2 externalizes real-time statistics” on page 1057 for information on when DB2 externalizes the statistics.

## Contents of the real-time statistics tables

The SYSIBM.TABLESPACESTATS table contains statistics information on table spaces and table space partitions. The SYSIBM.INDEXSPACESTATS table contains statistics information on index spaces and index space partitions.

Table 190 describes the columns of the TABLESPACESTATS table and explains how you can use them in deciding when to run REORG, RUNSTATS, or COPY.

*Table 190. Descriptions of columns in the TABLESPACESTATS table*

| Column name | Data type | Description |
|---|---|---|
| DBNAME | CHAR(8) NOT NULL | The name of the database.<br>This column is used to map a database to its statistics. |
| NAME | CHAR(8) NOT NULL | The name of the table space.<br>This column is used to map a table space to its statistics. |
| PARTITION | SMALLINT NOT NULL | The data set number within the table space.<br>This column is used to map a data set number in a table space to its statistics. For partitioned table spaces, this value corresponds to the partition number for a single partition. For nonpartitioned table spaces, this value is 0. |
| DBID | SMALLINT NOT NULL | The internal identifier of the database.<br>This column is used to map a DBID to its statistics. |
| PSID | SMALLINT NOT NULL | The internal identifier of the table space page set descriptor.<br>This column is used to map a PSID to its statistics. |

*Table 190. Descriptions of columns in the TABLESPACESTATS table (continued)*

| Column name | Data type | Description |
|---|---|---|
| UPDATESTATSTIME | TIMESTAMP NOT NULL WITH DEFAULT | The timestamp when the row was inserted or last updated.<br>This column is updated with the current timestamp when a row in the TABLESPACESTATS table is inserted or updated. You can use this column in several ways:<br>• To determine the actions that caused the latest change to the table. Do this by selecting any of the timestamp columns and comparing them to the UPDATESTATSTIME column.<br>• To determine whether an analysis of data is needed. This determination might be based on a given time interval, or on a combination of the time interval and the amount of activity.<br>For example, suppose you want to analyze statistics for the last seven days. To determine whether there has been any activity in the past seven days, check whether the difference between the current date and the UPDATESTATSTIME value is less than or equal to seven:<br>`(JULIAN_DAY(CURRENT DATE)-JULIAN_DAY(UPDATESTATSTIME))<= 7` |
| TOTALROWS | FLOAT | The number of rows or LOBs in the table space or partition.<br>If the table space contains more than one table, this value is the sum of all rows in all tables. A null value means that the number of rows is unknown, or REORG or LOAD has never been run.<br>Use this value with the value of any column that contains a number of affected rows to determine the percentage of rows that are affected by a particular action. |
| NACTIVE | INTEGER | The number of active pages in the table space or partition.<br>A null value means the number of active pages is unknown.<br>This value is equivalent to the number of preformatted pages. For multi-piece table spaces, this value is the total number of preformatted pages in all data sets.<br>Use this value with the value of any column that contains a number of affected pages to determine the percentage of pages that are affected by a particular action.<br>For example, suppose that your site's maintenance policies require that COPY is run after 20 per cent of the pages in a table space have changed. To determine if a COPY might be required, calculate the ratio of updated pages since the last COPY to the total number of active pages. If the percentage is greater than 20, you need to run COPY:<br>`((COPYUPDATEDPAGES*100)/NACTIVE)>20` |
| SPACE | INTEGER | The amount of space that is allocated to the table space or partition, in kilobytes.<br>For multi-piece linear page sets, this value is the amount of space in all data sets. A null value means the amount of space is unknown.<br>Use this value to monitor growth and validate design assumptions. |

*Table 190. Descriptions of columns in the TABLESPACESTATS table (continued)*

| Column name | Data type | Description |
|---|---|---|
| EXTENTS | SMALLINT | The number of physical extents in the table space or partition.<br>For multi-piece linear page sets, this value is the number of extents for the last data set. A null value means the number of extents is unknown.<br>Use this value to determine:<br>• When the primary or secondary allocation value for a table space or partition needs to be altered.<br>• When you are approaching the maximum number of extents and risking extend failures. |
| LOADRLASTTIME | TIMESTAMP | The timestamp of the last LOAD REPLACE on the table space or partition.<br>A null value means LOAD REPLACE has never been run on the table space or partition, or the timestamp of the last LOAD REPLACE is unknown.<br>You can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last LOAD REPLACE is more recent than the last COPY, you might need to run COPY:<br>`(JULIAN_DAY(LOADRLASTTIME)>JULIAN_DAY(COPYLASTTIME))` |
| REORGLASTTIME | TIMESTAMP | The timestamp of the last REORG on the table space or partition.<br>A null value means REORG has never been run on the table space or partition, or the timestamp of the last REORG is unknown.<br>You can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last REORG is more recent than the last COPY, you might need to run COPY:<br>`(JULIAN_DAY(REORGLASTTIME)>JULIAN_DAY(COPYLASTTIME))` |
| REORGINSERTS | INTEGER | The number of records or LOBs that have been inserted since the last REORG or LOAD REPLACE on the table space or partition.<br>A null value means that the number of inserted records or LOBs is unknown. |
| REORGDELETES | INTEGER | The number of records or LOBs that have been deleted since the last REORG or LOAD REPLACE on the table space or partition.<br>A null value means that the number of deleted records or LOBs is unknown. |

*Table 190. Descriptions of columns in the TABLESPACESTATS table (continued)*

| Column name | Data type | Description |
|---|---|---|
| REORGUPDATES | INTEGER | The number of rows that have been updated since the last REORG or LOAD REPLACE on the table space or partition.<br>This value does not include LOB updates because LOB updates are really deletions followed by insertions. A null value means that the number of updated rows is unknown.<br>This value can be used with REORGDELETES and REORGINSERTS to determine if a REORG is necessary. For example, suppose that your site's maintenance policies require that REORG is run after 20 per cent of the rows in a table space have changed. To determine if a REORG is required, calculate the sum of updated, inserted, and deleted rows since the last REORG. Then calculate the ratio of that sum to the total number of rows. If the percentage is greater than 20, you might need to run REORG:<br>`(((REORGINSERTS+REORGDELETES+REORGUPDATES)*100)/TOTALROWS)>20` |
| REORGDISORGLOB | INTEGER | The number of LOBs that were inserted since the last REORG or LOAD REPLACE that are not perfectly chunked.<br>A LOB is perfectly chunked if the allocated pages are in the minimum number of chunks. A null value means that the number of imperfectly chunked LOBs is unknown.<br>Use this value to determine whether you need to run REORG. For example, you might want to run REORG if the ratio of REORGDISORGLOB to the total number of LOBs is greater than 10 per cent:<br>`((REORGDISORGLOB*100)/TOTALROWS)>10` |
| REORGUNCLUSTINS | INTEGER | The number of records that were inserted since the last REORG or LOAD REPLACE. that are not well-clustered with respect to the clustering index.<br>A record is well-clustered if the record is inserted into a page that is within 16 pages of the ideal candidate page. The clustering index determines the ideal candidate page. A null value means that the number of badly-clustered pages is unknown.<br>You can use this value to determine whether you need to run REORG. For example, you might want to run REORG if the following comparison is true:<br>`((REORGUNCLUSTINS*100)/TOTALROWS)>10` |
| REORGMASSDELETE | INTEGER | The number of mass deletes from a segmented or LOB table space, or the number of dropped tables from a segmented table space, since the last REORG or LOAD REPLACE.<br>A null value means that the number of mass deletes is unknown.<br>If this value is non-zero, a REORG might be necessary. |
| REORGNEARINDREF | INTEGER | The number of overflow records that were created since the last REORG or LOAD REPLACE and were relocated near the pointer record.<br>For nonsegmented table spaces, a page is near the present page if the two page numbers differ by 16 or less. For segmented table spaces, a page is near the present page if the two page numbers differ by SEGSIZE*2 or less. A null value means that the number of overflow records near the pointer record is unknown. |

*Table 190. Descriptions of columns in the TABLESPACESTATS table  (continued)*

| Column name | Data type | Description |
|---|---|---|
| REORGFARINDEF | INTEGER | The number of overflow records that were created since the last REORG or LOAD REPLACE and were relocated far from the pointer record.<br>For nonsegmented table spaces, a page is far from the present page if the two page numbers differ by more than 16. For segmented table spaces, a page is far from the present page if the two page numbers differ by at least (SEGSIZE*2)+1. A null value means that the number of overflow records far from the pointer record is unknown.<br>For example, in a non-data sharing environment, you might run REORG if the following comparison is true:<br>`(((REORGNEARINDREF+REORGFARINDREF)*100)/TOTALROWS)>10`<br>In a data sharing environment, you might run REORG if the following comparison is true:<br>`(((REORGNEARINDREF+REORGFARINDREF)*100)/TOTALROWS)>5` |
| STATSLASTTIME | TIMESTAMP | The timestamp of the last RUNSTATS on the table space or partition.<br>A null value means RUNSTATS has never been run on the table space or partition, or the timestamp of the last RUNSTATS is unknown.<br>You can compare this timestamp to the timestamp of the last REORG on the same object to determine when RUNSTATS is needed. If the date of the last REORG is more recent than the last RUNSTATS, you might need to run RUNSTATS:<br>`(JULIAN_DAY(REORGLASTTIME)>JULIAN_DAY(STATSLASTTIME))` |
| STATSINSERTS | INTEGER | The number of records or LOBs that have been inserted since the last RUNSTATS on the table space or partition.<br>A null value means that the number of inserted records or LOBs is unknown. |
| STATSDELETES | INTEGER | The number of records or LOBs that have been deleted since the last RUNSTATS on the table space or partition.<br>A null value means that the number of deleted records or LOBs is unknown. |
| STATSUPDATES | INTEGER | The number of rows that have been updated since the last RUNSTATS on the table space or partition.<br>This value does not include LOB updates because LOB updates are really deletions followed by insertions. A null value means that the number of updated rows is unknown.<br>This value can be used with STATSDELETES and STATSINSERTS to determine if RUNSTATS is necessary. For example, suppose that your site's maintenance policies require that RUNSTATS is run after 20 per cent of the rows in a table space have changed. To determine if RUNSTATS is required, calculate the sum of updated, inserted, and deleted rows since the last RUNSTATS. Then calculate the ratio of that sum to the total number of rows. If the percentage is greater than 20, you need to run RUNSTATS:<br>`(((STATSINSERTS+STATSDELETES+STATSUPDATES)*100)/TOTALROWS)>20` |

*Table 190. Descriptions of columns in the TABLESPACESTATS table (continued)*

| Column name | Data type | Description |
|---|---|---|
| STATSMASSDELETE | INTEGER | The number of mass deletes from a segmented or LOB table space, or the number of dropped tables from a segmented table space, since the last RUNSTATS.<br>A null value means that the number of mass deletes is unknown.<br>If this value is non-zero, RUNSTATS might be necessary. |
| COPYLASTTIME | TIMESTAMP | The timestamp of the last full or incremental image copy on the table space or partition.<br>A null value means COPY has never been run on the table space or partition, or the timestamp of the last full image copy is unknown.<br>You can compare this timestamp to the timestamp of the last REORG on the same object to determine when a COPY is needed. If the date of the last REORG is more recent than the last COPY, you might need to run COPY:<br>`(JULIAN_DAY(REORGRLASTTIME)>JULIAN_DAY(COPYLASTTIME))` |
| COPYUPDATEDPAGES | INTEGER | The number of distinct pages that have been updated since the last COPY.<br>A null value means that the number of updated pages is unknown.<br>You can compare this value to the total number of pages to determine when a COPY is needed.<br>For example, you might want to take an incremental image copy when one percent of the pages have changed:<br>`((COPYUPDATEDPAGES*100)/NACTIVE)>1`<br>You might want to take a full image copy when 20 percent of the pages have changed:<br>`((COPYUPDATEDPAGES*100)/NACTIVE)>20` |
| COPYCHANGES | INTEGER | The number of insert, delete, and update operations since the last COPY.<br>A null value means that the number of insert, delete, or update operations is unknown.<br>This number indicates the approximate number of log records that DB2 processes to recover to the current state.<br>For example, you might want to take an incremental image copy when DB2 processes more than one percent of the rows from the logs:<br>`((COPYCHANGES*100)/TOTALROWS)>1`<br>You might want to take a full image copy when DB2 processes more than 10 percent of the rows from the logs:<br>`((COPYCHANGES*100)/TOTALROWS)>10` |

*Table 190. Descriptions of columns in the TABLESPACESTATS table (continued)*

| Column name | Data type | Description |
|---|---|---|
| COPYUPDATELRSN | CHAR(6) FOR BIT DATA | The LRSN or RBA of the first update after the last COPY.<br>A null value means that the LRSN or RBA is unknown.<br>Consider running COPY if this value is not in the active logs. To determine the oldest LRSN or RBA in the active logs, use the Print Log Map utility (DSNJU004). |
| COPYUPDATETIME | TIMESTAMP | The timestamp of the first update after the last COPY.<br>A null value means that the timestamp is unknown.<br>This value has a similar purpose to COPYUPDATELRSN. |

Table 191 describes the columns of the INDEXSPACESTATS table and explains how you can use them in deciding when to run REORG, RUNSTATS, or COPY.

*Table 191. Descriptions of columns in the INDEXSPACESTATS table*

| Column name | Data type | Description |
|---|---|---|
| DBNAME | CHAR(8) NOT NULL | The name of the database.<br>This column is used to map a database to its statistics. |
| NAME | CHAR(8) NOT NULL | The name of the index space.<br>This column is used to map an index space to its statistics. |
| PARTITION | SMALLINT NOT NULL | The data set number within the index space.<br>This column is used to map a data set number in an index space to its statistics. For partitioned index spaces, this value corresponds to the partition number for a single partition. For nonpartitioned index spaces, this value is 0. |
| DBID | SMALLINT NOT NULL | The internal identifier of the database.<br>This column is used to map a DBID to its statistics. |
| ISOBID | SMALLINT NOT NULL | The internal identifier of the index space page set descriptor.<br>This column is used to map an ISOBID to its statistics. |
| PSID | SMALLINT NOT NULL | The internal identifier of the table space page set descriptor for the table space on which the index that is represented by this row is created.<br>This column is used to map a PSID to the statistics for the associated index. |

*Table 191. Descriptions of columns in the INDEXSPACESTATS table (continued)*

| Column name | Data type | Description |
|---|---|---|
| UPDATESTATSTIME | TIMESTAMP NOT NULL WITH DEFAULT | The timestamp when the row was inserted or last updated.<br>This column is updated with the current timestamp when a row in the INDEXSPACESTATS table is inserted or updated. You can use this column in several ways:<br>• To determine the actions that caused the latest change to the INDEXSPACESTATS table. Do this by selecting any of the timestamp columns and comparing them to the UPDATESTATSTIME column.<br>• To determine whether an analysis of data is needed. This determination might be based on a given time interval, or on a combination of the time interval and the amount of activity.<br>For example, suppose you want to analyze statistics for the last seven days. To determine whether there has been any activity in the past seven days, check whether the difference between the current date and the UPDATESTATSTIME value is less than or equal to seven:<br>`(JULIAN_DAY(CURRENT DATE)-JULIAN_DAY(UPDATESTATSTIME))<= 7` |
| TOTALENTRIES | FLOAT | The number of entries, including duplicate entries, in the index space or partition.<br>A null value means that the number of entries is unknown, or REORG, LOAD, or REBUILD has never been run.<br>Use this value with the value of any column that contains a number of affected index entries to determine the percentage of index entries that are affected by a particular action. |
| NLEVELS | SMALLINT | The number of levels in the index tree.<br>A null value means that the number of levels is unknown. |
| NACTIVE | INTEGER | The number of active pages in the index space or partition.<br>A null value means the number of active pages is unknown.<br>This value is equivalent to the number of preformatted pages.<br>Use this value with the value of any column that contains a number of affected pages to determine the percentage of pages that are affected by a particular action.<br>For example, suppose that your site's maintenance policies require that COPY is run after 20 per cent of the pages in an index space have changed. To determine if a COPY is required, calculate the ratio of updated pages since the last COPY to the total number of active pages. If the percentage is greater than 20, you need to run COPY:<br>`((COPYUPDATEDPAGES*100)/NACTIVE)>20` |
| SPACE | INTEGER | The amount of space that is allocated to the index space or partition, in kilobytes.<br>For multi-piece linear page sets, this value is the amount of space in all data sets. A null value means the amount of space is unknown.<br>Use this value to monitor growth and validate design assumptions. |

*Table 191. Descriptions of columns in the INDEXSPACESTATS table (continued)*

| Column name | Data type | Description |
|---|---|---|
| EXTENTS | SMALLINT | The number of physical extents in the index space or partition.<br>For multi-piece linear page sets, this value is the number of extents for the last data set. A null value means the number of extents is unknown.<br>Use this value to determine:<br>• When the primary allocation value for an index space or partition needs to be altered.<br>• When you are approaching the maximum number of extents and risking extend failures. |
| LOADRLASTTIME | TIMESTAMP | The timestamp of the last LOAD REPLACE on the index space or partition.<br>A null value means that the timestamp of the last LOAD REPLACE is unknown.<br>If COPY YES was specified when the index was created (the value of COPY is Y in SYSIBM.SYSINDEXES), you can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last LOAD REPLACE is more recent than the last COPY, you might need to run COPY:<br>`(JULIAN_DAY(LOADRLASTTIME)>JULIAN_DAY(COPYLASTTIME))` |
| REBUILDLASTTIME | TIMESTAMP | The timestamp of the last REBUILD INDEX on the index space or partition.<br>A null value means the timestamp of the last REBUILD INDEX is unknown.<br>If COPY YES was specified when the index was created (the value of COPY is Y in SYSIBM.SYSINDEXES), you can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last REBUILD INDEX is more recent than the last COPY, you might need to run COPY:<br>`(JULIAN_DAY(REBUILDLASTTIME)>JULIAN_DAY(COPYLASTTIME))` |
| REORGLASTTIME | TIMESTAMP | The timestamp of the last REORG INDEX on the index space or partition.<br>A null value means the timestamp of the last REORG INDEX is unknown.<br>If COPY YES was specified when the index was created (the value of COPY is Y in SYSIBM.SYSINDEXES), you can compare this timestamp to the timestamp of the last COPY on the same object to determine when a COPY is needed. If the date of the last REORG INDEX is more recent than the last COPY, you might need to run COPY:<br>`(JULIAN_DAY(REORGLASTTIME)>JULIAN_DAY(COPYLASTTIME))` |
| REORGINSERTS | INTEGER | The number of index entries that have been inserted since the last REORG, REBUILD INDEX or LOAD REPLACE on the index space or partition.<br>A null value means that the number of inserted index entries is unknown. |

*Table 191. Descriptions of columns in the INDEXSPACESTATS table (continued)*

| Column name | Data type | Description |
|---|---|---|
| REORGDELETES | INTEGER | The number of index entries that have been deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE on the index space or partition.<br>A null value means that the number of deleted index entries is unknown.<br>This value can be used with REORGINSERTS to determine if a REORG is necessary. For example, suppose that your site's maintenance policies require that REORG is run after 20 per cent of the index entries have changed. To determine if a REORG is required, calculate the sum of inserted and deleted rows since the last REORG. Then calculate the ratio of that sum to the total number of index entries. If the percentage is greater than 20, you need to run REORG:<br>`(((REORGINSERTS+REORGDELETES)*100)/TOTALENTRIES)>20` |
| REORGAPPENDINSERT | INTEGER | The number of index entries that have been inserted since the last REORG, REBUILD INDEX or LOAD REPLACE on the index space or partition that have a key value that is greater than the maximum key value in the index or partition.<br>A null value means the number of inserted index entries is unknown.<br>This value can be used with REORGINSERTS to decide when to adjust the PCTFREE specification for the index. For example, if the ratio of REORGAPPENDINSERT to REORGINSERTS is greater than 10 per cent, you might need to run ALTER INDEX to adjust PCTFREE or run REORG more frequently:<br>`((REORGAPPENDINSERT*100)/REORGINSERTS)>10` |
| REORGPSEUDODELETES | INTEGER | The number of index entries that have been pseudo-deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE on the index space or partition. A pseudo-delete is a RID entry that has been marked as deleted.<br>A null value means that the number of pseudo-deleted index entries is unknown.<br>This value can be used to determine if a REORG is necessary. For example, if the ratio of pseudo-deletes to total index entries is greater than 10 per cent, you might need to run REORG:<br>`((REORGPSEUDODELETES*100)/TOTALENTRIES)>10` |
| REORGMASSDELETE | INTEGER | The number of times that an index or index space partition was mass deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE.<br>A null value means that the number of mass deletes is unknown.<br>If this value is non-zero, a REORG might be necessary. |
| REORGLEAFNEAR | INTEGER | The number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was near the location of the original page.<br>The higher part of a split page is near the original page if the two page numbers differ by 16 or less. A null value means that the number of split pages near their original pages is unknown. |

*Table 191. Descriptions of columns in the INDEXSPACESTATS table (continued)*

| Column name | Data type | Description |
|---|---|---|
| REORGLEAFFAR | INTEGER | The number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was far from the location of the original page.<br>The higher part of a split page is far from the original page if the two page numbers differ by more than 16. A null value means that the number of split pages that are far from their original pages is unknown.<br>This value can be used to decide when to run REORG. For example, calculate the ratio of index page splits in which the higher part of the split page was far from the location of the original page to the number of active pages. If this value is greater than 10 per cent, you might need to run REORG:<br>`((REORGLEAFFAR*100)/NACTIVE)>10` |
| REORGNUMLEVELS | INTEGER | The number of levels in the index tree that were added or removed since the last REORG, REBUILD INDEX, or LOAD REPLACE.<br>A null value means that the number of added or deleted levels is unknown.<br>If this value has increased since the last REORG, REBUILD INDEX, or LOAD REPLACE, you need to check other values such as REORGPSEUDODELETES to determine whether to run REORG.<br>If this value is less than zero, the index space contains empty pages. Running REORG can save disk space and decrease index sequential scan I/O time by eliminating those empty pages. |
| STATSLASTTIME | TIMESTAMP | The timestamp of the last RUNSTATS on the index space or partition.<br>A null value means RUNSTATS has never been run on the index space or partition, or the timestamp of the last RUNSTATS is unknown.<br>You can compare this timestamp to the timestamp of the last REORG on the same object to determine when RUNSTATS is needed. If the date of the last REORG is more recent than the last RUNSTATS, you might need to run RUNSTATS:<br>`(JULIAN_DAY(REORGLASTTIME)>JULIAN_DAY(STATSLASTTIME))` |
| STATSINSERTS | INTEGER | The number index entries that have been inserted since the last RUNSTATS on the index space or partition.<br>A null value means that the number of inserted index entries unknown. |

*Table 191. Descriptions of columns in the INDEXSPACESTATS table (continued)*

| Column name | Data type | Description |
|---|---|---|
| STATSDELETES | INTEGER | The number of index entries that have been deleted since the last RUNSTATS on the index space or partition.<br>A null value means that the number of deleted index entries unknown.<br>This value can be used with STATSINSERTS to determine if RUNSTATS is necessary. For example, suppose that your site's maintenance policies require that RUNSTATS is run after 20 per cent of the rows in an index space have changed. To determine if RUNSTATS is required, calculate the sum of inserted and deleted index entries since the last RUNSTATS. Then calculate the ratio of that sum to the total number of index entries. If the percentage is greater than 20, you need to run RUNSTATS:<br>`(((STATSINSERTS+STATSDELETES)*100)/TOTALENTRIES)>20` |
| STATSMASSDELETE | INTEGER | The number of times that the index or index space partition was mass deleted since the last RUNSTATS.<br>A null value means that the number of mass deletes is unknown.<br>If this value is non-zero, RUNSTATS might be necessary. |
| COPYLASTTIME | TIMESTAMP | The timestamp of the last full image copy on the index space or partition.<br>A null value means COPY has never been run on the index space or partition, or the timestamp of the last full image copy is unknown.<br>You can compare this timestamp to the timestamp of the last REORG on the same object to determine when a COPY is needed. If the date of the last REORG is more recent than the last COPY, you might need to run COPY:<br>`(JULIAN_DAY(REORGRLASTTIME)>JULIAN_DAY(COPYLASTTIME))` |
| COPYUPDATEDPAGES | INTEGER | The number of distinct pages that have been updated since the last COPY.<br>A null value means that the number of updated pages is unknown, or the index was created with COPY NO.<br>You can compare this value to the total number of pages to determine when a COPY is needed.<br>For example, you might want to take a full image copy when 20 percent of the pages have changed:<br>`((COPYUPDATEDPAGES*100)/NACTIVE)>20` |
| COPYCHANGES | INTEGER | The number of insert delete operations since the last COPY.<br>A null value means that the number of insert or update operations is unknown, or the index was created with COPY NO.<br>This number indicates the approximate number of log records that DB2 processes to recover to the current state.<br>For example, you might want to take a full image copy when DB2 processes more than 10 percent of the index entries from the logs:<br>`((COPYCHANGES*100)/TOTALENTRIES)>10` |

*Table 191. Descriptions of columns in the INDEXSPACESTATS table (continued)*

| Column name | Data type | Description |
|---|---|---|
| COPYUPDATELRSN | CHAR(6) FOR BIT DATA | The LRSN or RBA of the first update after the last COPY.<br>A null value means that the LRSN or RBA is unknown, or the index was created with COPY NO.<br>Consider running COPY if this value is not in the active logs. To determine the oldest LRSN or RBA in the active logs, use the Print Log Map utility (DSNJU004). |
| COPYUPDATETIME | TIMESTAMP | The timestamp of the first update after the last COPY.<br>A null value means that the timestamp is unknown, or the index was created with COPY NO.<br>This value has a similar purpose to COPYUPDATELRSN. |

# Operating with real-time statistics

To use the real-time statistics effectively, you need to understand when DB2 collects and externalizes them, and what factors in your system can affect the statistics. This section contains the following topics:

- “When DB2 externalizes real-time statistics”
- “How DB2 utilities affect the real-time statistics” on page 1058
- “How non-DB2 utilities affect real-time statistics” on page 1064
- “Real-time statistics on objects in work file databases and the TEMP database” on page 1065
- “Real-time statistics on read-only objects” on page 1065
- “How dropping objects affects real-time statistics” on page 1065
- “How SQL operations affect real-time statistics counters” on page 1065
- “Real-time statistics in data sharing” on page 1066
- “Improving concurrency with real-time statistics” on page 1066
- “Recovering the real-time statistics tables” on page 1066
- “Statistics accuracy” on page 1066

## When DB2 externalizes real-time statistics

DB2 externalizes real-time statistics at the following times:

- When you issue -STOP DATABASE(DSNRTSDB)

  This command stops the in-memory statistics database and externalizes statistics for all objects in the subsystem.
- When you issue -STOP DATABASE(*database-name*) SPACENAM(*space-name*)

  This command externalizes statistics only for *database-name* and *space-name*.
- At the end of the time interval that you specify during installation

  See “Setting the interval for writing real-time statistics” on page 1044 for information on how to set this time interval.
- When you issue -STOP DB2 MODE(QUIESCE)

  DB2 writes any statistics that are in memory when you issue this command to the statistics tables. However, if you issue -STOP DB2 MODE(FORCE), DB2 does not write the statistics, and you lose them.
- During utility operations

  “How DB2 utilities affect the real-time statistics” on page 1058 gives details on how the utilities modify the statistics tables.

  DB2 does not maintain real-time statistics for any objects in the real-time statistics database. Therefore, if you run a utility with a utility list, and the list contains any real-time statistics objects, DB2 does not externalize real-time statistics during the execution of that utility for any of the objects in the utility list.

***Recommendation:*** Do not include real-time statistics objects in utility lists.

DB2 does not externalize real-time statistics at a tracker site.

## How DB2 utilities affect the real-time statistics

In general, SQL INSERT, UPDATE, and DELETE statements cause DB2 to modify the real-time statistics. However, certain DB2 utilities also affect the statistics. The following section discuss the affect of each of those utilities on the statistics.

### How LOAD affects real-time statistics

Table 192 shows how running LOAD on a table space or table space partition affects the TABLESPACESTATS statistics.

*Table 192. Changed TABLESPACESTATS values during LOAD*

| Column name | Settings for LOAD REPLACE after RELOAD phase |
|---|---|
| TOTALROWS | Number of rows or LOBs loaded[3] |
| NACTIVE | Actual value |
| SPACE | Actual value |
| EXTENTS | Actual value |
| LOADRLASTTIME | Current timestamp |
| REORGINSERTS | 0 |
| REORGDELETES | 0 |
| REORGUPDATES | 0 |
| REORGDISORGLOB | 0 |
| REORGUNCLUSTINS | 0 |
| REORGMASSDELETE | 0 |
| REORGNEARINDREF | 0 |
| REORGFARINDEF | 0 |
| STATSLASTTIME | Current timestamp[1] |
| STATSINSERTS | 0[1] |
| STATSDELETES | 0[1] |
| STATSUPDATES | 0[1] |
| STATSMASSDELETE | 0[1] |
| COPYLASTTIME | Current timestamp[2] |
| COPYUPDATEDPAGES | 0[2] |
| COPYCHANGES | 0[2] |
| COPYUPDATELRSN | Null[2] |
| COPYUPDATETIME | Null[2] |

*Table 192. Changed TABLESPACESTATS values during LOAD (continued)*

| Column name | Settings for LOAD REPLACE after RELOAD phase |
|---|---|

**Note:**

1. DB2 sets this value only if the LOAD invocation includes the STATISTICS option.
2. DB2 sets this value only if the LOAD invocation includes the COPYDDN option.
3. Under certain conditions, such as a utility restart, the LOAD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null. Some rows that are loaded into a table space and are included in this value might later be removed during the index validation phase or the referential integrity check. DB2 includes counts of those removed records in the statistics that record deleted records.

Table 193 shows how running LOAD affects the INDEXSPACESTATS statistics for an index space or physical index partition.

*Table 193. Changed INDEXSPACESTATS values during LOAD*

| Column name | Settings for LOAD REPLACE after BUILD phase |
|---|---|
| TOTALENTRIES | Number of index entries added[3] |
| NLEVELS | Actual value |
| NACTIVE | Actual value |
| SPACE | Actual value |
| EXTENTS | Actual value |
| LOADRLASTTIME | Current timestamp |
| REORGINSERTS | 0 |
| REORGDELETES | 0 |
| REORGAPPENDINSERT | 0 |
| REORGPSEUDODELETES | 0 |
| REORGMASSDELETE | 0 |
| REORGLEAFNEAR | 0 |
| REORGLEAFFAR | 0 |
| REORGNUMLEVELS | 0 |
| STATSLASTTIME | Current timestamp[1] |
| STATSINSERTS | 0[1] |
| STATSDELETES | 0[1] |
| STATSMASSDELETE | 0[1] |
| COPYLASTTIME | Current timestamp[2] |
| COPYUPDATEDPAGES | 0[2] |
| COPYCHANGES | 0[2] |
| COPYUPDATELRSN | Null[2] |
| COPYUPDATETIME | Null[2] |

**Notes:**

1. DB2 sets this value only if the LOAD invocation includes the STATISTICS option.
2. DB2 sets this value only if the LOAD invocation includes the COPYDDN option.
3. Under certain conditions, such as a utility restart, the LOAD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.

*For a logical index partition:*

- DB2 does not reset the nonpartitioning index when it does a LOAD REPLACE on a partition. Therefore, DB2 does not reset the statistics for the index. The REORG counters from the last REORG are still correct. DB2 updates LOADRLASTTIME when the entire nonpartitioning index is replaced.
- When DB2 does a LOAD RESUME YES on a partition, after the BUILD phase, DB2 increments TOTALENTRIES by the number of index entries that were inserted during the BUILD phase.

## How REORG affects real-time statistics

Table 194 shows how running REORG on a table space or table space partition affects the TABLESPACESTATS statistics.

*Table 194. Changed TABLESPACESTATS values during REORG*

| Column name | Settings for REORG SHRLEVEL NONE after RELOAD phase | Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase |
|---|---|---|
| TOTALROWS | Number rows or LOBs loaded[3] | For SHRLEVEL REFERENCE: Number of rows or LOBs loaded during RELOAD phase<br>For SHRLEVEL CHANGE: Number of rows or LOBs loaded during RELOAD phase +Number of rows inserted during LOG APPLY phase-Number of rows deleted during LOG phase |
| NACTIVE | Actual value | Actual value |
| SPACE | Actual value | Actual value |
| EXTENTS | Actual value | Actual value |
| REORGLASTTIME | Current timestamp | Current timestamp |
| REORGINSERTS | 0 | Actual value[4] |
| REORGDELETES | 0 | Actual value[4] |
| REORGUPDATES | 0 | Actual value[4] |
| REORGDISORGLOB | 0 | Actual value[4] |
| REORGUNCLUSTINS | 0 | Actual value[4] |
| REORGMASSDELETE | 0 | Actual value[4] |
| REORGNEARINDREF | 0 | Actual value[4] |
| REORGFARINDEF | 0 | Actual value[4] |
| STATSLASTTIME | Current timestamp[1] | Current timestamp[1] |
| STATSINSERTS | 0[1] | Actual value[4] |
| STATSDELETES | 0[1] | Actual value[4] |
| STATSUPDATES | 0[1] | Actual value[4] |
| STATSMASSDELETE | 0[1] | Actual value[4] |
| COPYLASTTIME | Current timestamp[2] | Current timestamp |
| COPYUPDATEDPAGES | 0[2] | Actual value[4] |
| COPYCHANGES | 0[2] | Actual value[4] |
| COPYUPDATELRSN | Null[2] | Actual value[5] |

*Table 194. Changed TABLESPACESTATS values during REORG (continued)*

| Column name | Settings for REORG SHRLEVEL NONE after RELOAD phase | Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase |
|---|---|---|
| COPYUPDATETIME | Null[2] | Actual value[5] |

**Notes:**

1. DB2 sets this value only if the REORG invocation includes the STATISTICS option.
2. DB2 sets this value only if the REORG invocation includes the COPYDDN option.
3. Under certain conditions, such as a utility restart, the REORG utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null. Some rows that are loaded into a table space and are included in this value might later be removed during the index validation phase or the referential integrity check. DB2 includes counts of those removed records in the statistics that record deleted records.
4. This is the actual number of inserts, updates, or deletes that are due to applying the log to the shadow copy.
5. This is the LRSN or timestamp for the first update that is due to applying the log to the shadow copy.

Table 195 shows how running REORG affects the INDEXSPACESTATS statistics for an index space or physical index partition.

*Table 195. Changed INDEXSPACESTATS values during REORG*

| Column name | Settings for REORG SHRLEVEL NONE after RELOAD phase | Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase |
|---|---|---|
| TOTALENTRIES | Number of index entries added[3] | For SHRLEVEL REFERENCE: Number of index entries added during BUILD phase<br>For SHRLEVEL CHANGE: Number of index entries added during BUILD phase +Number of index entries added during LOG phase-Number of index entries deleted during LOG phase |
| NLEVELS | Actual value | Actual value |
| NACTIVE | Actual value | Actual value |
| SPACE | Actual value | Actual value |
| EXTENTS | Actual value | Actual value |
| REORGLASTTIME | Current timestamp | Current timestamp |
| REORGINSERTS | 0 | Actual value[4] |
| REORGDELETES | 0 | Actual value[4] |
| REORGAPPENDINSERT | 0 | Actual value[4] |
| REORGPSEUDODELETES | 0 | Actual value[4] |
| REORGMASSDELETE | 0 | Actual value[4] |
| REORGLEAFNEAR | 0 | Actual value[4] |
| REORGLEAFFAR | 0 | Actual value[4] |
| REORGNUMLEVELS | 0 | Actual value[4] |
| STATSLASTTIME | Current timestamp[1] | Current timestamp[1] |
| STATSINSERTS | 0[1] | Actual value[4] |
| STATSDELETES | 0[1] | Actual value[4] |
| STATSMASSDELETE | 0[1] | Actual value[4] |
| COPYLASTTIME | Current timestamp[2] | Unchanged[5] |

*Table 195. Changed INDEXSPACESTATS values during REORG (continued)*

| Column name | Settings for REORG SHRLEVEL NONE after RELOAD phase | Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase |
|---|---|---|
| COPYUPDATEDPAGES | 0[2] | Unchanged[5] |
| COPYCHANGES | 0[2] | Unchanged[5] |
| COPYUPDATELRSN | Null[2] | Unchanged[5] |
| COPYUPDATETIME | Null[2] | Unchanged[5] |

**Notes:**

1. DB2 sets this value only if the REORG invocation includes the STATISTICS option.
2. DB2 sets this value only if the REORG invocation includes the COPYDDN option.
3. Under certain conditions, such as a utility restart, the REORG utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.
4. This is the actual number of inserts, updates, or deletes that are due to applying the log to the shadow copy.
5. Inline COPY is not allowed for SHRLEVEL CHANGE or SHRLEVEL REFERENCE.

*For a logical index partition:* DB2 does not reset the nonpartitioning index when it does a REORG on a partition. Therefore, DB2 does not reset the statistics for the index. The REORG counters from the last REORG are still correct. DB2 updates REORGLASTTIME when the entire nonpartitioning index is reorganized.

## How REBUILD INDEX affects real-time statistics

Table 196 shows how running REBUILD INDEX affects the INDEXSPACESTATS statistics for an index space or physical index partition.

*Table 196. Changed INDEXSPACESTATS values during REBUILD INDEX*

| Column name | Settings after BUILD phase |
|---|---|
| TOTALENTRIES | Number of index entries added[1] |
| NLEVELS | Actual value |
| NACTIVE | Actual value |
| SPACE | Actual value |
| EXTENTS | Actual value |
| REBUILDLASTTIME | Current timestamp |
| REORGINSERTS | 0 |
| REORGDELETES | 0 |
| REORGAPPENDINSERT | 0 |
| REORGPSEUDODELETES | 0 |
| REORGMASSDELETE | 0 |
| REORGLEAFNEAR | 0 |
| REORGLEAFFAR | 0 |
| REORGNUMLEVELS | 0 |

**Note:**

1. Under certain conditions, such as a utility restart, the REBUILD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.

*For a logical index partition:* DB2 does not collect TOTALENTRIES statistics for the entire nonpartitioning index when it runs REBUILD INDEX. Therefore, DB2 does not reset the statistics for the index. The REORG counters from the last REORG are still correct. DB2 updates REBUILDLASTTIME when the entire nonpartitioning index is rebuilt.

## How RUNSTATS affects real-time statistics

Only RUNSTATS UPDATE ALL affects the real-time statistics. When the RUNSTATS job starts, DB2 externalizes all in-memory statistics to the real-time statistics tables.

Table 197 shows how running RUNSTATS UPDATE ALL on a table space or table space partition affects the TABLESPACESTATS statistics.

*Table 197. Changed TABLESPACESTATS values during RUNSTATS UPDATE ALL*

| Column name | During UTILINIT phase | After RUNSTATS phase |
|---|---|---|
| STATSLASTTIME | Current timestamp[1] | Timestamp of the start of RUNSTATS phase |
| STATSINSERTS | Actual value[1] | Actual value[2] |
| STATSDELETES | Actual value[1] | Actual value[2] |
| STATSUPDATES | Actual value[1] | Actual value[2] |
| STATSMASSDELETE | Actual value[1] | Actual value[2] |

**Notes:**

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

Table 198 shows how running RUNSTATS UPDATE ALL on an index affects the INDEXSPACESTATS statistics.

*Table 198. Changed INDEXSPACESTATS values during RUNSTATS UPDATE ALL*

| Column name | During UTILINIT phase | After RUNSTATS phase |
|---|---|---|
| STATSLASTTIME | Current timestamp[1] | Timestamp of the start of RUNSTATS phase |
| STATSINSERTS | Actual value[1] | Actual value[2] |
| STATSDELETES | Actual value[1] | Actual value[2] |
| STATSMASSDELETE | Actual value[1] | Actual value[2] |

**Notes:**

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

## How COPY affects real-time statistics

When a COPY job starts, DB2 externalizes all in-memory statistics to the real-time statistics tables. Statistics are gathered only for a full image copy or an incremental copy, but not for a data set copy.

Table 199 on page 1064 shows how running COPY on a table space or table space partition affects the TABLESPACESTATS statistics.

*Table 199. Changed TABLESPACESTATS values during COPY*

| Column name | During UTILINIT phase | After COPY phase |
|---|---|---|
| COPYLASTTIME | Current timestamp[1] | Timestamp of the start of COPY phase |
| COPYUPDATEDPAGES | Actual value[1] | Actual value[2] |
| COPYCHANGES | Actual value[1] | Actual value[2] |
| COPYUPDATELRSN | Actual value[1] | Actual value[3] |
| COPYUPDATETIME | Actual value[1] | Actual value[3] |

**Notes:**

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.
3. This value is null for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

Table 200 shows how running COPY on an index affects the INDEXSPACESTATS statistics.

*Table 200. Changed INDEXSPACESTATS values during COPY*

| Column name | During UTILINIT phase | After COPY phase |
|---|---|---|
| COPYLASTTIME | Current timestamp[1] | Timestamp of the start of COPY phase |
| COPYUPDATEDPAGES | Actual value[1] | Actual value[2] |
| COPYCHANGES | Actual value[1] | Actual value[2] |
| COPYUPDATELRSN | Actual value[1] | Actual value[3] |
| COPYUPDATETIME | Actual value[1] | Actual value[3] |

**Note:**

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.
3. This value is null for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

### How RECOVER affects real-time statistics

After recovery to the current state, the in-memory counter fields are still valid, so DB2 does not modify them. However, after a point-in-time recovery, the statistics might not be valid. DB2 therefore sets all the REORG, STATS, and COPY counter statistics to null after a point-in-time recovery. After recovery to the current state, DB2 sets NACTIVE, SPACE, and EXTENTS to their new values. After a point-in-time recovery, DB2 sets NLEVELS, NACTIVE, SPACE, and EXTENTS to their new values.

## How non-DB2 utilities affect real-time statistics

Non-DB2 utilities do not affect real-time statistics. Therefore, an object that is the target of a non-DB2 COPY, LOAD, REBUILD, REORG or RUNSTATS can cause incorrect statistics to be inserted in the real-time statistics tables. Follow this process to ensure correct statistics when you run non-DB2 utilities:

1. Stop the table space or index on which you plan to run the utility. This action causes DB2 to write the in-memory statistics to the real-time statistics tables and initialize the in-memory counters.
2. Run the utility.
3. When the utility completes, update the statistics tables with new totals, timestamps and zero incremental counter values.

## Real-time statistics on objects in work file databases and the TEMP database

Although you cannot run utilities on objects in the work files databases and TEMP database, DB2 records the NACTIVE, SPACE, and EXTENTS statistics on table spaces in those databases.

## Real-time statistics on read-only objects

DB2 does not externalize the NACTIVE, SPACE, or EXTENTS statistics for read-only objects.

## How dropping objects affects real-time statistics

If you drop a table space or index, DB2 deletes its statistics from the real-time statistics tables. However, if the real-time statistics database is not available when you drop a table space or index, the statistics remain in the real-time statistics tables, even though the corresponding object no longer exists. You need to use SQL DELETE statements to manually remove those rows from the real-time statistics tables.

If a row still exists in the real-time statistics tables for a dropped table space or index, and if you create a new object with the same DBID and PSID as the dropped object, DB2 reinitializes the row before it updates any values in that row.

## How SQL operations affect real-time statistics counters

SQL operations affect the counter columns in the real-time statistics tables. These are the columns that record that record the number of insert, delete, or update operations, as well as total counters TOTALROWS and TOTALENTRIES.

***UPDATE:*** When you perform an UPDATE, DB2 increments the update counters.

***INSERT:*** When you perform an INSERT, DB2 increments the insert counters. DB2 keeps separate counters for clustered and unclustered UPDATEs.

***DELETE:*** When you perform a DELETE, DB2 increments the delete counters.

***ROLLBACK:*** When you perform a ROLLBACK, DB2 increments the the counters, depending on the type of SQL operation that is rolled back:

| Rolled-back SQL statement | Incremented counters |
|---|---|
| UPDATE | Update counters |
| INSERT | Delete counters |
| DELETE | Insert counters |

***Mass DELETE:*** When you perform a mass delete operation on a table space does not cause DB2 to reset the counter columns in the real-time statistics tables. After a mass delete operation, the value in a counter column includes the count from before the mass delete operation, as well as the count after the mass delete operation.

## Real-time statistics in data sharing

In a data sharing environment, DB2 members update their statistics serially. Each member reads the target row from the statistics table, obtains a lock, aggregates its in-memory statistics, and updates the statistics table with the new totals. Each member sets its own interval for writing real-time statistics.

DB2 does locking based on the lock size of the DSNRTSDB.DSNRTSTS table space. DB2 uses cursor stability isolation and CURRENTDATA(YES) when it reads the statistics tables.

At the beginning of a RUNSTATS job, all data sharing members externalize their statistics to the real-time statistics tables and reset their in-memory statistics. If all members cannot externalize their statistics, DB2 sets STATSLASTTIME to null. An error in gathering and externalizing statistics does not prevent RUNSTATS from running.

At the beginning of a COPY job, all data sharing members externalize their statistics to the real-time statistics tables and reset their in-memory statistics. If all members cannot externalize their statistics, DB2 sets COPYLASTTIME to null. An error in gathering and externalizing statistics does not prevent COPY from running.

Utilities that reset page sets to empty can invalidate the in-memory statistics of other DB2 members. The member that resets a page set notifies the other DB2 members that a page set has been reset to empty, and the in-memory stats are invalidated. If the notify process fails, the utility that resets the page set does not fail. DB2 sets the appropriate timestamp (REORGLASTTIME, STATSLASTTIME or COPYLASTTIME) to null in the row for the empty page set to indicate that the statistics for that page set are unknown.

## Improving concurrency with real-time statistics

Follow these recommendations to reduce the risk of timeouts and deadlocks when you work with the real-time statistics tables:

- When you run COPY, RUNSTATS or REORG on the real-time statistics objects, use SHRLEVEL CHANGE.
- When you execute SQL statements to query the real-time statistics tables, use uncommitted read isolation.

## Recovering the real-time statistics tables

When you recover a DB2 subsystem after a disaster, you need to perform the following actions on the real-time statistics database:

- Recover the real-time statistics objects after you recover the DB2 catalog and directory.
- Start the real-time statistics database explicitly, after DB2 restart.

## Statistics accuracy

In general, the real-time statistics are accurate values. However, several factors can affect the accuracy of the statistics:

- Certain utility restart scenarios
- A DB2 subsystem failure

- A notify failure in a data sharing environment

If you think that some statistics values might be inaccurate, you can correct the statistics by running REORG, RUNSTATS, or COPY on the objects for which DB2 generated the statistics.

# Appendix H. Stored procedures shipped with DB2

DB2 provides several stored procedures that you can call in your application programs to perform a number of utility and application programming 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 Appendix B of *DB2 Utility Guide and Reference* 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 Appendix B of *DB2 Utility Guide and Reference* 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 Appendix B of *DB2 Utility Guide and Reference* for information.
- The real-time statistics stored procedure (DSNACCOR)

  This stored procedure queries the DB2 real-time statistics tables to help you determine when you should run COPY, REORG, or RUNSTATS, or enlarge your DB2 data sets. See “The DB2 real-time statistics stored procedure” for information.
- The WLM environment refresh stored procedure (WLM_REFRESH)

  This stored procedure lets you refresh a WLM environment from a remote workstation. See Appendix I of *DB2 Application Programming and SQL Guide* for information.
- The CICS transaction invocation stored procedure (DSNACICS)

  This stored procedure lets you invoke CICS transactions fom a remote workstation. See “The CICS transaction invocation stored procedure (DSNACICS)” on page 1087 for information.

## The DB2 real-time statistics stored procedure

The information under this heading is Product-sensitive Programming Interface and Associated Guidance Information, as defined in “Notices” on page 1095.

The DSNACCOR stored procedure is a sample stored procedure that makes recommendations to help you maintain your DB2 databases. In particular, DSNACCOR performs these actions:

- Recommends when you should reorganize, image copy, or update statistics for table spaces or index spaces
- Indicates table spaces or index spaces that have exceeded their data set extents
- Indicates whether an object for which it recommends an action is in a restricted state when DSNACCOR runs

DSNACCOR uses data from the SYSIBM.TABLESPACESTATS and SYSIBM.INDEXSPACESTATS real-time statistics tables to make its recommendations. DSNACCOR provides its recommendations in a result set.

DSNACCOR uses the set of criteria shown in “Formulas for recommending actions” on page 1077 to evaluate table spaces and index spaces. By default, DSNACCOR evaluates *all* table spaces and index spaces in the subsystem that have entries in the real-time statistics tables. However, you can override this default through input parameters.

***Important information about DSNACCOR recommendations:***

- DSNACCOR makes recommendations based on general formulas that require input from the user about the maintenance policies for a subsystem. These recommendations might not be accurate for every installation.
- If the real-time statistics tables contain information for only a small percentage of your DB2 subsystem, the recommendations that DSNACCOR makes might not be accurate for the entire subsystem.
- Before you perform any action that DSNACCOR recommends, ensure that the object for which DSNACCOR makes the recommendation is available, and that it is possible to perform the recommended action on that object. For example, before you can perform an image copy on an index, the index must have the COPY YES attribute.

## Environment

DSNACCOR must run in a WLM-established stored procedure address space.

DSNACCOR creates and uses declared temporary tables. Therefore, before you can invoke DSNACCOR, you need to create a TEMP database and segmented table spaces in the TEMP database. Specify a 4KB buffer pool when you create the TEMP database. For information on creating TEMP databases and table spaces, see the CREATE DATABASE and CREATE TABLESPACE sections in Chapter 5 of *DB2 SQL Reference*.

Before you can invoke DSNACCOR, the real-time statistics tables, SYSIBM.TABLESPACESTATS and SYSIBM.INDEXSPACESTATS, must exist, and the real-time statistics database must be started. See “Appendix G. Real-time statistics tables” on page 1043 for information on the real-time statistics tables.

## Authorization required

To execute the CALL DSNACCOR statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNACCOR
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

The owner of the package or plan that contains the CALL statement must also have:

- SELECT authority on the real-time statistics tables
- The DISPLAY system privilege

## DSNACCOR syntax diagram

The following syntax diagram shows the CALL statement for invoking DSNACCOR. Because the linkage convention for DSNACCOR is GENERAL WITH NULLS, if you pass parameters in host variables, you need to include a null indicator with every host variable. Null indicators for input host variables must be initialized before you execute the CALL statement.

```
►►──CALL──DSNACCOR──(──┬─QueryType──┬─,──┬─ObjectType──┬─,──┬─ICType──┬─,──┬─StatsSchema──┬─,──────────►
                       └─NULL───────┘    └─NULL────────┘    └─NULL────┘    └─NULL─────────┘

►──┬─CatlgSchema──┬─,──┬─LocalSchema──┬─,──┬─ChkLvl──┬─,──┬─Criteria──┬─,──┬─Unused──┬─,──────────────►
   └─NULL─────────┘    └─NULL─────────┘    └─NULL────┘    └─NULL──────┘    └─NULL────┘

►──┬─CRUpdatedPagesPct──┬─,──┬─CRChangesPct──┬─,──┬─CRDaySncLastCopy──┬─,──┬─ICRUpdatedPagesPct──┬─,──►
   └─NULL───────────────┘    └─NULL──────────┘    └─NULL──────────────┘    └─NULL────────────────┘

►──┬─ICRChangesPct──┬─,──┬─CRIndexSize──┬─,───────────────────────────────────────────────────────────►
   └─NULL───────────┘    └─NULL─────────┘

►──┬─RRTInsDelUpdPct──┬─,──┬─RRTUnclustInsPct──┬─,──┬─RRTDisorgLOBPct──┬─,──┬─RRTMassDelLimit──┬─,────►
   └─NULL─────────────┘    └─NULL──────────────┘    └─NULL─────────────┘    └─NULL─────────────┘

►──┬─RRTIndRefLimit──┬─,──┬─RRIInsertDeletePct──┬─,──┬─RRIAppendInsertPct──┬─,────────────────────────►
   └─NULL────────────┘    └─NULL────────────────┘    └─NULL────────────────┘

►──┬─RRIPseudoDeletePct──┬─,──┬─RRIMassDelLimit──┬─,──┬─RRILeafLimit──┬─,──┬─RRINumLevelsLimit──┬─,───►
   └─NULL────────────────┘    └─NULL─────────────┘    └─NULL──────────┘    └─NULL───────────────┘

►──┬─SRTInsDelUpdPct──┬─,──┬─SRTInsDelUpdAbs──┬─,──┬─SRTMassDelLimit──┬─,──┬─SRIInsDelUpdPct──┬─,────►
   └─NULL─────────────┘    └─NULL─────────────┘    └─NULL─────────────┘    └─NULL─────────────┘

►──┬─SRIInsDelUpdAbs──┬─,──┬─SRIMassDelLimit──┬─,──┬─ExtentLimit──┬─,────────────────────────────────►
   └─NULL─────────────┘    └─NULL─────────────┘    └─NULL─────────┘

►──LastStatement,──ReturnCode,──ErrorMessage,──IFCARetCode,──IFCAResCode,──XSBytes──)──────────────►◄
```

# DSNACCOR option descriptions

In the following option descriptions, the default value for an input parameter is the value that DSNACCOR uses if you specify a null value.

*QueryType*

Specifies the types of actions that DSNACCOR recommends. This field contains one or more of the following values. Each value is enclosed in single quotation marks and separated from other values by a space.

| | |
|---|---|
| **ALL** | Make recommendations for all of the following actions. |
| **COPY** | Make a recommendation on whether to perform an image copy. |
| **RUNSTATS** | Make a recommendation on whether to perform RUNSTATS. |
| **REORG** | Make a recommendation on whether to perform REORG. Choosing this value causes DSNACCOR to process the EXTENTS value also. |
| **EXTENTS** | Indicate when data sets have exceeded a user-specified extents limit. |
| **RESTRICT** | Indicate which objects are in a restricted state. |

This is an input parameter of type VARCHAR(40). The default is 'ALL'.

*ObjectType*

Specifies the types of objects for which DSNACCOR recommends actions:

**ALL** Table spaces and index spaces.

**TS** Table spaces only.

**IX** Index spaces only.

This is an input parameter of type VARCHAR(3). The default is 'ALL'.

*ICType*

Specifies the types of image copies for which DSNACCOR should make recommendations:

**F** Full image copy.

**I** Incremental image copy. This value is valid for table spaces only.

**B** Full image copy or incremental image copy.

This is an input parameter of type VARCHAR(1). The default is 'B'.

*StatsSchema*

Specifies the qualifier for the real-time statistics table names. This is an input parameter of type VARCHAR(128). The default is 'SYSIBM'.

*CatlgSchema*

Specifies the qualifier for DB2 catalog table names. This is an input parameter of type VARCHAR(128). The default is 'SYSIBM'.

*LocalSchema*

Specifies the qualifier for the names of tables that DSNACCOR creates. This is an input parameter of type VARCHAR(128). The default is 'DSNACC'.

*ChkLvl*

Specifies the types of checking that DSNACCOR performs, and indicates whether to include objects that fail those checks in the DSNACCOR recommendations result set. This value is the sum of any combination of the following values:

**0** DSNACCOR performs none of the following actions.

**1** For objects that are listed in the recommendations result set, check the SYSTABLESPACE or SYSINDEXES catalog tables to ensure that those objects have not been deleted. If value 16 is *not* also chosen, exclude rows for the deleted objects from the recommendations result set.

**2** For index spaces that are listed in the recommendations result set, check the SYSTABLES, SYSTABLESPACE, and SYSINDEXES catalog tables to determine the name of the table space that is associated with each index space.

Choosing this value causes DSNACCOR to also check for rows in the recommendations result set for objects that have been deleted but have entries in the real-time statistics tables (value 1). This means that if value 16 is *not* also chosen, rows for deleted objects are excluded from the recommendations result set.

**4** Check whether rows that are in the DSNACCOR recommendations result set refer to objects that are in the exception table. For recommendations result set rows that have corresponding exception

#	table rows, copy the contents of the QUERYTYPE column of the exception table to the INEXCEPTTABLE column of the recommendations result set.

#	**8**	Check whether objects that have rows in the recommendations result set are restricted. Indicate the restricted status in the OBJECTSTATUS column of the result set.

#	**16**	For objects that are listed in the recommendations result set, check the SYSTABLESPACE or SYSINDEXES catalog tables to ensure that those objects have not been deleted (value 1). In result set rows for deleted objects, specify the word ORPHANED in the OBJECTSTATUS column.

#	**32**	Exclude rows from the DSNACCOR recommendations result set for index spaces for which the related table spaces have been recommended for REORG. Choosing this value causes DSNACCOR to perform the actions for values 1 and 2.

#	This is an input parameter of type INTEGER. The default is 7 (values 1+2+4).

#	*Criteria*
Narrows the set of objects for which DSNACCOR makes recommendations. This value is the search condition of an SQL WHERE clause. This is an input parameter of type VARCHAR(4096). The default is that DSNACCOR makes recommendations for all table spaces and index spaces in the subsystem.

#	*Unused*
A parameter that is reserved for future use. Specify the null value for this parameter. This is an input parameter of type VARCHAR(80).

#	*CRUpdatedPagesPct*
Specifies a criterion for recommending a full image copy on a table space or index space. For a table space, if the ratio of distinct updated pages to preformatted pages, expressed as a percentage, is greater than this value, DSNACCOR recommends an image copy. (See item 2 in Figure 153 on page 1078.) For an index space, if the ratio of distinct updated pages to preformatted pages, expressed as a percentage, is greater than this value, *and* the number of active pages in the index space or partition is greater than *CRIndexSize*, DSNACCOR recommends an image copy. (See items 2 and 3 in Figure 154 on page 1078.) This is an input parameter of type INTEGER. The default is 20.

#	*CRChangesPct*
Specifies a criterion for recommending a full image copy on a table space or index space. For a table space, if the ratio of the number INSERTs, UPDATEs, and DELETEs since the last image copy to the total number of rows or LOBs in a table space or partition, expressed as a percentage, is greater than this value, DSNACCOR recommends an image copy. (See item 3 in Figure 153 on page 1078.) For an index space, if the ratio of the number INSERTs and DELETEs since the last image copy to the total number of entries in the index space or partition, expressed as a percentage, is greater than this value, *and* the number of active pages in the index space or partition is greater than *CRIndexSize*, DSNACCOR recommends an image copy. (See items 2 and 4 in Figure 154 on page 1078.) This is an input parameter of type INTEGER. The default is 10.

#	*CRDaySncLastCopy*
Specifies a criterion for recommending a full image copy on a table space or index space. If the number of days since the last image copy is greater than this value, DSNACCOR recommends an image copy. (See item 1 in Figure 153 on page 1078

on page 1078 and item 1 in Figure 154 on page 1078.) This is an input parameter of type INTEGER. The default is 7.

*ICRUpdatedPagesPct*
Specifies a criterion for recommending an incremental image copy on a table space. If the ratio of the number of distinct pages updated since the last image copy to the total number of active pages in the table space or partition, expressed as a percentage, is greater than this value, DSNACCOR recommends an incremental image copy. (See item 1 in Figure 155 on page 1078.) This is an input parameter of type INTEGER. The default is 1.

*ICRChangesPct*
Specifies a criterion for recommending an incremental image copy on a table space. If the ratio of the number INSERTs, UPDATEs, and DELETEs since the last image copy to the total number of rows or LOBs in a table space or partition, expressed as a percentage, is greater than this value, DSNACCOR recommends an incremental image copy. (See item 2 in Figure 155 on page 1078.) This is an input parameter of type INTEGER. The default is 1.

*CRIndexSize*
Combined with *CRUpdatedPagesPct* or *CRChangesPct*, specifies a criterion for recommending a full image copy on an index space. (See items 2, 3, and 4 in Figure 154 on page 1078.) This is an input parameter of type INTEGER. The default is 50.

*RRTInsDelUpdPct*
Specifies a criterion for recommending that the REORG utility should be run on a table space. If the ratio of the sum of INSERTs, UPDATEs, and DELETEs since the last REORG to the total number of rows or LOBs in the table space or partition, expressed as a percentage, is greater than this value, DSNACCOR recommends running REORG. (See item 1 in Figure 156 on page 1078.) This is an input parameter of type INTEGER. The default is 20.

*RRTUnclustInsPct*
Specifies a criterion for recommending that the REORG utility should be run on a table space. If the ratio of the number of unclustered INSERTs to the total number of rows or LOBs in the table space or partition, expressed as a percentage, is greater than this value, DSNACCOR recommends running REORG. (See item 2 in Figure 156 on page 1078.) This is an input parameter of type INTEGER. The default is 10.

*RRTDisorgLOBPct*
Specifies a criterion for recommending that the REORG utility should be run on a table space. If the ratio of the number of imperfectly chunked LOBs to the total number of rows or LOBs in the table space or partition, expressed as a percentage, is greater than this value, DSNACCOR recommends running REORG. (See item 3 in Figure 156 on page 1078.) This is an input parameter of type INTEGER. The default is 10.

*RRTMassDelLimit*
Specifies a criterion for recommending that the REORG utility should be run on a table space. If the number of mass deletes from a segmented or LOB table space since the last REORG or LOAD REPLACE, or the number of dropped tables from a nonsegmented table space since the last REORG or LOAD REPLACE is greater than this value, DSNACCOR recommends running REORG. (See item 5 in Figure 156 on page 1078.) This is an input parameter of type INTEGER. The default is 0.

*RRTIndRefLimit*
Specifies a criterion for recommending that the REORG utility should be run on

a table space. If the ratio of the total number of overflow records that were created since the last REORG or LOAD REPLACE to the total number of rows or LOBs in the table space or partition, expressed as a percentage, is greater than this value, DSNACCOR recommends running REORG. (See item 4 in Figure 156 on page 1078.) This is an input parameter of type INTEGER. The default is 10.

*RRIInsertDeletePct*

Specifies a criterion for recommending that the REORG utility should be run on an index space. If the ratio of the sum of the number of index entries that were inserted and deleted since the last REORG to the total number of index entries in the index space or partition, expressed as a percentage, is greater than this value, DSNACCOR recommends running REORG. (See item 1 in Figure 157 on page 1079.) This is an input parameter of type INTEGER. The default is 20.

*RRIAppendInsertPct*

Specifies a criterion for recommending that the REORG utility should be run on an index space. If the ratio of the number of index entries that were inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE, and had a key value greater than the maximum key value in the index space or partition, to the number of index entries in the index space or partition, expressed as a percentage, is greater than this value, DSNACCOR recommends running REORG. (See item 2 in Figure 157 on page 1079.) This is an input parameter of type INTEGER. The default is 10.

*RRIPseudoDeletePct*

Specifies a criterion for recommending that the REORG utility should be run on an index space. If the ratio of the number of index entries that were pseudo-deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE to the number of index entries in the index space or partition, expressed as a percentage, is greater than this value, DSNACCOR recommends running REORG. (See item 3 in Figure 157 on page 1079.) This is an input parameter of type INTEGER. The default is 10.

*RRIMassDelLimit*

Specifies a criterion for recommending that the REORG utility should be run on an index space. If the number of mass deletes from an index space or partition since the last REORG, REBUILD, or LOAD REPLACE is greater than this value, DSNACCOR recommends running REORG. (See item 4 in Figure 157 on page 1079.) This is an input parameter of type INTEGER. The default is 0.

*RRILeafLimit*

Specifies a criterion for recommending that the REORG utility should be run on an index space. If the ratio of the number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was far from the location of the original page, to the total number of active pages in the index space or partition, expressed as a percentage, is greater than this value, DSNACCOR recommends running REORG. (See item 5 in Figure 157 on page 1079.) This is an input parameter of type INTEGER. The default is 10.

*RRINumLevelsLimit*

Specifies a criterion for recommending that the REORG utility should be run on an index space. If the number of levels in the index tree that were added or removed since the last REORG, REBUILD INDEX, or LOAD REPLACE is greater than this value, DSNACCOR recommends running REORG. (See item 6 in Figure 157 on page 1079.) This is an input parameter of type INTEGER. The default is 0.

*SRTInsDelUpdPct*

Combined with *SRTInsDelUpdAbs*, specifies a criterion for recommending that the RUNSTATS utility should be run on a table space. If the ratio of the number INSERTs, UPDATEs, and DELETEs since the last RUNSTATS on a table space or partition, to the total number of rows or LOBs in the table space or partition, expressed as a percentage, is greater than *SRTInsDelUpdPct*, *and* the sum of the number INSERTs, UPDATEs, and DELETEs since the last RUNSTATS on a table space or partition is greater than *SRTInsDelUpdAbs*, DSNACCOR recommends running RUNSTATS. (See items 1 and 2 in Figure 158 on page 1079.) This is an input parameter of type INTEGER. The default is 20.

*SRTInsDelUpdAbs*

Combined with *SRTInsDelUpdAbs*, specifies a criterion for recommending that the RUNSTATS utility should be run on a table space. (See items 1 and 2 in Figure 158 on page 1079.) This is an input parameter of type INTEGER. The default is 0.

*SRTMassDelLimit*

Specifies a criterion for recommending that the RUNSTATS utility should be run on a table space. If the number of mass deletes from a table space or partition since the last REORG or LOAD REPLACE is greater than this value, DSNACCOR recommends running RUNSTATS. (See item 3 in Figure 158 on page 1079 .) This is an input parameter of type INTEGER. The default is 0.

*SRIInsDelUpdPct*

Combined with *SRIInsDelUpdAbs*, specifies a criterion for recommending that the RUNSTATS utility should be run on an index space. If the ratio of the number inserted and deleted index entries since the last RUNSTATS on an index space or partition, to the total number of index entries in the index space or partition, expressed as a percentage, is greater than *SRIInsDelUpdPct*, *and* the sum of the number inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRIInsDelUpdAbs*, DSNACCOR recommends running RUNSTATS. (See items 1 and 2 in Figure 159 on page 1079.) This is an input parameter of type INTEGER. The default is 20.

*SRIInsDelUpdAbs*

Combined with *SRIInsDelUpdPct*, specifies a criterion for recommending that the RUNSTATS utility should be run on an index space. (See items 1 and 2 in Figure 159 on page 1079.) This is an input parameter of type INTEGER. The default is 0.

*SRIMassDelLimit*

Specifies a criterion for recommending that the RUNSTATS utility should be run on an index space. If the number of mass deletes from an index space or partition since the last REORG, REBUILD INDEX, or LOAD REPLACE is greater than this value, DSNACCOR recommends running RUNSTATS. (See item 3 in Figure 159 on page 1079.) This is an input parameter of type INTEGER. The default is 0.

*ExtentLimit*

Specifies a criterion for recommending that the RUNSTATS or REORG utility should be run on a table space or index space. Also specifies that DSNACCOR should warn the user that the table space or index space has used too many extents. If the number of physical extents in the index space, table space, or partition is greater than this value, DSNACCOR recommends running RUNSTATS or REORG and altering data set allocations. (See Figure 160 on page 1079.) This is an input parameter of type INTEGER. The default is 50.

*LastStatement*
When DSNACCOR returns a severe error (return code 12), this field contains the SQL statement that was executing when the error occurred. This is an output parameter of type VARCHAR(8012).

*ReturnCode*
The return code from DSNACCOR execution. Possible values are:

**0** DSNACCOR executed successfully. The *ErrorMsg* parameter contains the approximate percentage of the total number of objects in the subsystem that have information in the real-time statistics tables.

**4** DSNACCOR completed, but one or more input parameters might be incompatible. The *ErrorMsg* parameter contains the input parameters that might be incompatible.

**8** DSNACCOR terminated with errors. The *ErrorMsg* parameter contains a message that describes the error.

**12** DSNACCOR terminated with severe errors. The *ErrorMsg* parameter contains a message that describes the error. The *LastStatement* parameter contains the SQL statement that was executing when the error occurred.

**14** DSNACCOR terminated because it could not access one or more of the real-time statistics tables. The *ErrorMsg* parameter contains the names of the tables that DSNACCOR could not access.

**15** DSNACCOR terminated because it encountered a problem with one of the declared temporary tables that it defines and uses.

**16** DSNACCOR terminated because it could not define a declared temporary table. No table spaces were defined in the TEMP database.

**NULL** DSNACCOR terminated but could not set a return code.

This is an output parameter of type INTEGER.

*ErrorMsg*
Contains information about DSNACCOR execution. If DSNACCOR runs successfully (*ReturnCode*=0), this field contains the approximate percentage of objects in the subsystem that are in the real-time statistics tables. Otherwise, this field contains error messages. This is an output parameter of type VARCHAR(1331).

*IFCARetCode*
Contains the return code from an IFI COMMAND call. DSNACCOR issues commands through the IFI interface to determine the status of objects. This is an output parameter of type INTEGER.

*IFCAResCode*
Contains the reason code from an IFI COMMAND call. This is an output parameter of type INTEGER.

*XSBytes*
Contains the number of bytes of information that did not fit in the IFI return area after an IFI COMMAND call. This is an output parameter of type INTEGER.

## Formulas for recommending actions

The following formulas specify the criteria that DSNACCOR uses for its recommendations and warnings. The variables in italics are DSNACCOR input parameters. The capitalized variables are columns of the

SYSIBM.TABLESPACESTATS or SYSIBM.INDEXSPACESTATS tables. The numbers to the right of selected items are reference numbers for the option descriptions in the previous section.

```
((QueryType='COPY' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL') AND
 ICType='F') AND
 (COPYLASTTIME IS NULL OR
 REORGLASTTIME>COPYLASTTIME OR
 LOADRLASTTIME>COPYLASTTIME OR
 (CURRENT DATE-COPYLASTTIME)>CRDaySncLastCopy OR       1
 (COPYUPDATEDPAGES*100)/NACTIVE>CRUpdatedPagesPct OR   2
 (COPYCHANGES*100)/TOTALROWS>CRChangesPct)             3
```

*Figure 153. When DSNACCOR recommends a full image copy on a table space*

```
((QueryType='COPY' OR QueryType='ALL') AND
 (ObjectType='IX' OR ObjectType='ALL') AND
 (ICType='F'  OR ICType='B')) AND
 (COPYLASTTIME IS NULL OR
 REORGLASTTIME>COPYLASTTIME OR
 LOADRLASTTIME>COPYLASTTIME OR
 REBUILDLASTTIME>COPYLASTTIME OR
 (CURRENT DATE-COPYLASTTIME)>CRDaySncLastCopy OR        1
 (NACTIVE>CRIndexSize AND                               2
 ((COPYUPDATEDPAGES*100)/NACTIVE>CRUpdatedPagesPct OR   3
 (COPYCHANGES*100)/TOTALENTRIES>CRChangesPct)))         4
```

*Figure 154. When DSNACCOR recommends a full image copy on an index space*

```
((QueryType='COPY' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL') AND
 ICType='I' AND
 COPYLASTTIME IS NOT NULL) AND
 (LOADRLASTTIME>COPYLASTTIME OR
 REORGLASTTIME>COPYLASTTIME OR
 (COPYUPDATEDPAGES*100)/NACTIVE>ICRUpdatedPagesPct OR   1
 (COPYCHANGES*100)/TOTALROWS>ICRChangesPct))            2
```

*Figure 155. When DSNACCOR recommends an incremental image copy on a table space*

```
((QueryType='REORG' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL')) AND
 (REORGLASTTIME IS NULL OR
 ((REORGINSERTS+REORGDELETES+REORGUPDATES)*100)/TOTALROWS>RRTInsDelUpdPct OR   1
 (REORGUNCLUSTINS*100)/TOTALROWS>RRTUnclustInsPct OR                           2
 (REORGDISORGLOB*100)/TOTALROWS>RRTDisorgLOBPct OR                             3
 ((REORGNEARINDREF+REORGFARINDREF)*100)/TOTALROWS>RRTIndRefLimit OR            4
 REORGMASSDELETE>RRTMassDelLimit OR                                            5
 EXTENTS>ExtentLimit)                                                          6
```

*Figure 156. When DSNACCOR recommends a REORG on a table space*

```
((QueryType='REORG' OR QueryType='ALL') AND
 (ObjectType='IX' OR ObjectType='ALL')) AND
 (REORGLASTTIME IS NULL OR
 ((REORGINSERTS+REORGDELETES)*100)/TOTALENTRIES>RRIInsertDeletePct OR         1
 (REORGAPPENDINSERT*100)/TOTALENTRIES>RRIAppendInsertPct OR                   2
 (REORGPSEUDODELETES*100)/TOTALENTRIES>RRIPseudoDeletePct OR                  3
 REORGMASSDELETE>RRIMassDeleteLimit OR                                        4
 (REORGLEAFFAR*100)/NACTIVE>RRILeafLimit OR                                   5
 REORGNUMLEVELS>RRINumLevelsLimit OR                                          6
 EXTENTS>ExtentLimit)                                                         7
```

*Figure 157. When DSNACCOR recommends a REORG on an index space*

```
((QueryType='RUNSTATS' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL')) AND
 (STATSLASTTIME IS NULL OR
 (((STATSINSERTS+STATSDELETES+STATSUPDATES)*100)/TOTALROWS>SRTInsDelUpdPct AND  1
 (STATSINSERTS+STATSDELETES+STATSUPDATES)>SRTInsDelUpdAbs) OR                   2
 STATSMASSDELETE>SRTMassDeleteLimit)                                            3
```

*Figure 158. When DSNACCOR recommends RUNSTATS on a table space*

```
((QueryType='RUNSTATS' OR QueryType='ALL') AND
 (ObjectType='IX' OR ObjectType='ALL')) AND
 (STATSLASTTIME IS NULL OR
 (((STATSINSERTS+STATSDELETES)*100)/TOTALENTRIES>SRIInsDelUpdPct AND  1
 (STATSINSERTS+STATSDELETES)>SRIInsDelUpdAbs) OR                      2
 STATSMASSDELETE>SRIMassDelLimit)                                     3
```

*Figure 159. When DSNACCOR recommends RUNSTATS on an index space*

```
EXTENTS>ExtentLimit
```

*Figure 160. When DSNACCOR warns that too many data set extents for a table space or index space are used*

## Using an exception table

An exception table is an optional, user-created DB2 table that you can use to place information in the INEXCEPTTABLE column of the recommendations result set. You can put any information in the INEXCEPTTABLE column, but the most common use of this column is to filter the recommendations result set. Each row in the exception table represents an object for which you want to provide information for the recommendations result set.

To create the exception table, execute the following SQL statement:

```
CREATE TABLE DSNACC.EXCEPT_TBL
 (DBNAME CHAR(8) NOT NULL,
  NAME CHAR(8) NOT NULL,
  QUERYTYPE CHAR(40) NOT NULL);
```

The meanings of the columns are:

**DBNAME**
The database name for an object in the exception table.

**NAME**
The table space name or index space name for an object in the exception table.

**QUERYTYPE**
The information that you want to place in the INEXCEPTTABLE column of the recommendations result set.

If you put a null value in this column, DSNACCOR puts the value YES in the INEXCEPTTABLE column of the recommendations result set row for the object that matches the DBNAME and NAME values.

After you create the exception table, insert a row for each object for which you want to include information in the INEXCEPTTABLE column. For example, suppose that you want the INEXCEPTTABLE column to contain the string ’IRRELEVANT’ for table space STAFF in database DSNDB04. You also want the INEXCEPTTABLE column to contain ’CURRENT’ for table space DSN8S71D in database DSN8D71A. Execute these INSERT statements:

```
INSERT INTO DSNACC.EXCEPT_TBL VALUES('DSNDB04 ', 'STAFF ', 'IRRELEVANT');
INSERT INTO DSNACC.EXCEPT_TBL VALUES('DSN8D71A', 'DSN8S71D', 'CURRENT');
```

To use the contents of INEXCEPTTABLE for filtering, include a condition that involves the INEXCEPTTABLE column in the search condition that you specify in your *criteria* input parameter. For example, suppose that you want to include all rows for database DSNDB04 in the recommendations result set, except for those rows that contain the string ’IRRELEVANT’ in the INEXCEPTTABLE column. You might include the following search condition in your *criteria* input parameter:

```
DBNAME='DSNDB04' AND INEXCEPTTABLE<>'IRRELEVANT'
```

## Example of DSNACCOR invocation

The following COBOL example shows variable declarations and an SQL CALL for obtaining recommendations for objects in databases DSN8D71A and DSN8D71L.

```
 WORKING-STORAGE SECTION.
⋮

***********************
* DSNACCOR PARAMETERS *
***********************
 01  QUERYTYPE.
     49   QUERYTYPE-LN        PICTURE S9(4) COMP VALUE 40.
     49   QUERYTYPE-DTA       PICTURE X(40)  VALUE 'ALL'.
 01  OBJECTTYPE.
     49   OBJECTTYPE-LN       PICTURE S9(4) COMP VALUE 3.
     49   OBJECTTYPE-DTA      PICTURE X(3)  VALUE 'ALL'.
 01  ICTYPE.
     49   ICTYPE-LN           PICTURE S9(4) COMP VALUE 1.
     49   ICTYPE-DTA          PICTURE X(1)  VALUE 'B'.
 01  STATSSCHEMA.
     49   STATSSCHEMA-LN      PICTURE S9(4) COMP VALUE 128.
     49   STATSSCHEMA-DTA     PICTURE X(128)  VALUE 'SYSIBM'.
 01  CATLGSCHEMA.
 49   CATLGSCHEMA-LN      PICTURE S9(4) COMP VALUE 128.
     49   CATLGSCHEMA-DTA     PICTURE X(128)  VALUE 'SYSIBM'.
 01  LOCALSCHEMA.
     49   LOCALSCHEMA-LN      PICTURE S9(4) COMP VALUE 128.
     49   LOCALSCHEMA-DTA     PICTURE X(128)  VALUE 'DSNACC'.
 01  CHKLVL                   PICTURE S9(9) COMP VALUE +3.
 01  CRITERIA.
     49   CRITERIA-LN         PICTURE S9(4) COMP VALUE 4096.
```

```
#        49   CRITERIA-DTA        PICTURE X(4096)  VALUE SPACES.
#    01  RESTRICTED.
#        49   RESTRICTED-LN       PICTURE S9(4) COMP VALUE 80.
#        49   RESTRICTED-DTA      PICTURE X(80)  VALUE SPACES.
#    01  CRUPDATEDPAGESPCT        PICTURE S9(9) COMP VALUE +0.
#    01  CRCHANGESPCT             PICTURE S9(9) COMP VALUE +0.
#    01  CRDAYSNCLASTCOPY         PICTURE S9(9) COMP VALUE +0.
#    01  ICRUPDATEDPAGESPCT       PICTURE S9(9) COMP VALUE +0.
#    01  ICRCHANGESPCT            PICTURE S9(9) COMP VALUE +0.
#    01  CRINDEXSIZE              PICTURE S9(9) COMP VALUE +0.
#    01  RRTINSDELUPDPCT          PICTURE S9(9) COMP VALUE +0.
#    01  RRTUNCLUSTINSPCT         PICTURE S9(9) COMP VALUE +0.
#    01  RRTDISORGLOBPCT          PICTURE S9(9) COMP VALUE +0.
#    01  RRTMASSDELLIMIT          PICTURE S9(9) COMP VALUE +0.
#    01  RRTINDREFLIMIT           PICTURE S9(9) COMP VALUE +0.
#    01  RRIINSERTDELETEPCT       PICTURE S9(9) COMP VALUE +0.
#    01  RRIAPPENDINSERTPCT       PICTURE S9(9) COMP VALUE +0.
#    01  RRIPSEUDODELETEPCT       PICTURE S9(9) COMP VALUE +0.
#    01  RRIMASSDELLIMIT          PICTURE S9(9) COMP VALUE +0.
#    01  RRILEAFLIMIT             PICTURE S9(9) COMP VALUE +0.
#    01  RRINUMLEVELSLIMIT        PICTURE S9(9) COMP VALUE +0.
#    01  SRTINSDELUPDPCT          PICTURE S9(9) COMP VALUE +0.
#    01  SRTINSDELUPDABS          PICTURE S9(9) COMP VALUE +0.
#    01  SRTMASSDELLIMIT          PICTURE S9(9) COMP VALUE +0.
#    01  SRIINSDELUPDPCT          PICTURE S9(9) COMP VALUE +0.
#    01  SRIINSDELUPDABS          PICTURE S9(9) COMP VALUE +0.
#    01  SRIMASSDELLIMIT          PICTURE S9(9) COMP VALUE +0.
#    01  EXTENTLIMIT              PICTURE S9(9) COMP VALUE +0.
#    01  LASTSTATEMENT.
#        49   LASTSTATEMENT-LN    PICTURE S9(4) COMP VALUE 8012.
#        49   LASTSTATEMENT-DTA   PICTURE X(8012)  VALUE SPACES.
#    01  RETURNCODE               PICTURE S9(9) COMP VALUE +0.
#    01  ERRORMSG
#        49   ERRORMSG-LN         PICTURE S9(4) COMP VALUE 1331.
#        49   ERRORMSG-DTA        PICTURE X(1331)  VALUE SPACES.
#    01  IFCARETCODE              PICTURE S9(9) COMP VALUE +0.
#    01  IFCARESCODE              PICTURE S9(9) COMP VALUE +0.
#    01  XSBYTES                  PICTURE S9(9) COMP VALUE +0.
#   *****************************************
#   * INDICATOR VARIABLES.                  *
#   * INITIALIZE ALL NON-ESSENTIAL INPUT    *
#   * VARIABLES TO -1, TO INDICATE THAT THE *
#   * INPUT VALUE IS NULL.                  *
#   *****************************************
#    01  QUERYTYPE-IND            PICTURE S9(4) COMP-4 VALUE +0.
#    01  OBJECTTYPE-IND           PICTURE S9(4) COMP-4 VALUE +0.
#    01  ICTYPE-IND               PICTURE S9(4) COMP-4 VALUE +0.
#    01  STATSSCHEMA-IND          PICTURE S9(4) COMP-4 VALUE -1.
#    01  CATLGSCHEMA-IND          PICTURE S9(4) COMP-4 VALUE -1.
#    01  LOCALSCHEMA-IND          PICTURE S9(4) COMP-4 VALUE -1.
#    01  CHKLVL-IND               PICTURE S9(4) COMP-4 VALUE -1.
#    01  CRITERIA-IND             PICTURE S9(4) COMP-4 VALUE -1.
#    01  RESTRICTED-IND           PICTURE S9(4) COMP-4 VALUE -1.
#    01  CRUPDATEDPAGESPCT-IND    PICTURE S9(4) COMP-4 VALUE -1.
#    01  CRCHANGESPCT-IND         PICTURE S9(4) COMP-4 VALUE -1.
#    01  CRDAYSNCLASTCOPY-IND     PICTURE S9(4) COMP-4 VALUE -1.
#    01  ICRUPDATEDPAGESPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
#    01  ICRCHANGESPCT-IND        PICTURE S9(4) COMP-4 VALUE -1.
#    01  CRINDEXSIZE-IND          PICTURE S9(4) COMP-4 VALUE -1.
#    01  RRTINSDELUPDPCT-IND      PICTURE S9(4) COMP-4 VALUE -1.
#    01  RRTUNCLUSTINSPCT-IND     PICTURE S9(4) COMP-4 VALUE -1.
#    01  RRTDISORGLOBPCT-IND      PICTURE S9(4) COMP-4 VALUE -1.
#    01  RRTMASSDELLIMIT-IND      PICTURE S9(4) COMP-4 VALUE -1.
#    01  RRTINDREFLIMIT-IND       PICTURE S9(4) COMP-4 VALUE -1.
#    01  RRIINSERTDELETEPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
#    01  RRIAPPENDINSERTPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
#    01  RRIPSEUDODELETEPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
```

```
 01  RRIMASSDELLIMIT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  RRILEAFLIMIT-IND         PICTURE S9(4) COMP-4 VALUE -1.
 01  RRINUMLEVELSLIMIT-IND    PICTURE S9(4) COMP-4 VALUE -1.
 01  SRTINSDELUPDPCT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRTINSDELUPDABS-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRTMASSDELLIMIT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRIINSDELUPDPCT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRIINSDELUPDABS-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  SRIMASSDELLIMIT-IND      PICTURE S9(4) COMP-4 VALUE -1.
 01  EXTENTLIMIT-IND          PICTURE S9(4) COMP-4 VALUE -1.
 01  LASTSTATEMENT-IND        PICTURE S9(4) COMP-4 VALUE +0.
 01  RETURNCODE-IND           PICTURE S9(4) COMP-4 VALUE +0.
 01  ERRORMSG-IND             PICTURE S9(4) COMP-4 VALUE +0.
 01  IFCARETCODE-IND          PICTURE S9(4) COMP-4 VALUE +0.
 01  IFCARESCODE-IND          PICTURE S9(4) COMP-4 VALUE +0.
 01  XSBYTES-IND              PICTURE S9(4) COMP-4 VALUE +0.

 PROCEDURE DIVISION.
:
:

*********************************************************
* SET VALUES FOR DSNACCOR INPUT PARAMETERS:             *
* - USE THE CHKLVL PARAMETER TO CAUSE DSNACCOR TO CHECK *
*   FOR ORPHANED OBJECTS AND INDEX SPACES WITHOUT       *
*   TABLE SPACES, BUT INCLUDE THOSE OBJECTS IN THE      *
*   RECOMMENDATIONS RESULT SET (CHKLVL=1+2+16=19)       *
* - USE THE CRITERIA PARAMETER TO CAUSE DSNACCOR TO     *
*   MAKE RECOMMENDATIONS ONLY FOR OBJECTS IN DATABASES  *
*   DSN8D71A AND DSN8D71L.                              *
* - FOR THE FOLLOWING PARAMETERS, SET THESE VALUES,     *
*   WHICH ARE LOWER THAN THE DEFAULTS:                  *
*   CRUPDATEDPAGESPCT   4                               *
*   CRCHANGESPCT        2                               *
*   RRTINSDELUPDPCT     2                               *
*   RRTUNCLUSTINSPCT    5                               *
*   RRTDISORGLOBPCT     5                               *
*   RRIAPPENDINSERTPCT  5                               *
*   SRTINSDELUPDPCT     5                               *
*   SRIINSDELUPDPCT     5                               *
*   EXTENTLIMIT         3                               *
*********************************************************
     MOVE 19 TO CHKLVL.
     MOVE SPACES TO CRITERIA-DTA.
     MOVE 'DBNAME = ''DSN8D71A'' OR DBNAME = ''DSN8D71L'''
           TO CRITERIA-DTA.
     MOVE 46 TO CRITERIA-LN.
     MOVE 4 TO CRUPDATEDPAGESPCT.
     MOVE 2 TO CRCHANGESPCT.
     MOVE 2 TO RRTINSDELUPDPCT.
     MOVE 5 TO RRTUNCLUSTINSPCT.
     MOVE 5 TO RRTDISORGLOBPCT.
     MOVE 5 TO RRIAPPENDINSERTPCT.
     MOVE 5 TO SRTINSDELUPDPCT.
     MOVE 5 TO SRIINSDELUPDPCT.
     MOVE 3 TO EXTENTLIMIT.
********************************
* INITIALIZE OUTPUT PARAMETERS *
********************************
     MOVE SPACES TO LASTSTATEMENT-DTA.
     MOVE 1 TO LASTSTATEMENT-LN.
     MOVE 0 TO RETURNCODE-O2.
     MOVE SPACES TO ERRORMSG-DTA.
     MOVE 1 TO ERRORMSG-LN.
     MOVE 0 TO IFCARETCODE.
     MOVE 0 TO IFCARESCODE.
     MOVE 0 TO XSBYTES.
```

```
********************************************************
* SET THE INDICATOR VARIABLES TO 0 FOR NON-NULL INPUT *
* PARAMETERS (PARAMETERS FOR WHICH YOU DO NOT WANT    *
* DSNACCOR TO USE DEFAULT VALUES) AND FOR OUTPUT      *
* PARAMETERS.                                         *
********************************************************
     MOVE 0 TO CHKLVL-IND.
     MOVE 0 TO CRITERIA-IND.
     MOVE 0 TO CRUPDATEDPAGESPCT-IND.
     MOVE 0 TO CRCHANGESPCT-IND.
     MOVE 0 TO RRTINSDELUPDPCT-IND.
     MOVE 0 TO RRTUNCLUSTINSPCT-IND.
     MOVE 0 TO RRTDISORGLOBPCT-IND.
     MOVE 0 TO RRIAPPENDINSERTPCT-IND.
     MOVE 0 TO SRTINSDELUPDPCT-IND.
     MOVE 0 TO SRIINSDELUPDPCT-IND.
     MOVE 0 TO EXTENTLIMIT-IND.
     MOVE 0 TO LASTSTATEMENT-IND.
     MOVE 0 TO RETURNCODE-IND.
     MOVE 0 TO ERRORMSG-IND.
     MOVE 0 TO IFCARETCODE-IND.
     MOVE 0 TO IFCARESCODE-IND.
     MOVE 0 TO XSBYTES-IND.
⋮


*****************
* CALL DSNACCOR *
*****************
     EXEC SQL
      CALL SYSPROC.DSNACCOR
      (:QUERYTYPE           :QUERYTYPE-IND,
       :OBJECTTYPE          :OBJECTTYPE-IND,
       :ICTYPE              :ICTYPE-IND,
       :STATSSCHEMA         :STATSSCHEMA-IND,
       :CATLGSCHEMA         :CATLGSCHEMA-IND,
       :LOCALSCHEMA         :LOCALSCHEMA-IND,
       :CHKLVL              :CHKLVL-IND,
       :CRITERIA            :CRITERIA-IND,
       :CRUPDATEDPAGESPCT   :CRUPDATEDPAGESPCT-IND,
       :CRCHANGESPCT        :CRCHANGESPCT-IND,
       :CRDAYSNCLASTCOPY    :CRDAYSNCLASTCOPY-IND,
       :ICRUPDATEDPAGESPCT  :ICRUPDATEDPAGESPCT-IND,
       :CRINDEXSIZE         :CRINDEXSIZE-IND,
       :RRTINSDELUPDPCT     :RRTINSDELUPDPCT-IND,
       :RRTUNCLUSTINSPCT    :RRTUNCLUSTINSPCT-IND,
       :RRTDISORGLOBPCT     :RRTDISORGLOBPCT-IND,
       :RRTMASSDELLIMIT     :RRTMASSDELLIMIT-IND,
       :RRTINDREFLIMIT      :RRTINDREFLIMIT-IND,
       :RRIINSERTDELETEPCT  :RRIINSERTDELETEPCT-IND,
       :RRIAPPENDINSERTPCT  :RRIAPPENDINSERTPCT-IND,
       :RRIPSEUDODELETEPCT  :RRIPSEUDODELETEPCT-IND,
       :RRIMASSDELLIMIT     :RRIMASSDELLIMIT-IND,
       :RRILEAFLIMIT        :RRILEAFLIMIT-IND,
       :RRINUMLEVELSLIMIT   :RRINUMLEVELSLIMIT-IND,
       :SRTINSDELUPDPCT     :SRTINSDELUPDPCT-IND,
       :SRTINSDELUPDABS     :SRTINSDELUPDABS-IND,
       :SRTMASSDELLIMIT     :SRTMASSDELLIMIT-IND,
       :SRIINSDELUPDPCT     :SRIINSDELUPDPCT-IND,
       :SRIINSDELUPDABS     :SRIINSDELUPDABS-IND,
       :SRIMASSDELLIMIT     :SRIMASSDELLIMIT-IND,
       :EXTENTLIMIT         :EXTENTLIMIT-IND,
       :LASTSTATEMENT       :LASTSTATEMENT-IND,
       :RETURNCODE          :RETURNCODE-IND,
       :ERRORMSG            :ERRORMSG-IND,
       :IFCARETCODE         :IFCARETCODE-IND,
       :IFCARESCODE        :IFCARESCODE-IND,
       :XSBYTES            :XSBYTES-IND)
     END-EXEC.
```

## DSNACCOR output

If DSNACCOR executes successfully, in addition to the output parameters described in “DSNACCOR option descriptions” on page 1071, DSNACCOR returns two result sets.

The first result set contains the results from IFI COMMAND calls that DSNACCOR makes. Table 201 shows the format of the first result set.

*Table 201. Result set row for first DSNACCOR result set*

| Column name | Data type | Contents |
|---|---|---|
| RS_SEQUENCE | INTEGER | Sequence number of the output line |
| RS_DATA | CHAR(80) | A line of command output |

The second result set contains DSNACCOR's recommendations. This result set contains one or more rows for a table space or index space. A nonpartitioned table space or nonpartitioning index space can have at most one row in the result set. A partitioned table space or partitioning index space can have at most one row for each partition. A table space, index space, or partition has a row in the result set if the following conditions are true:

- If the *Criteria* input parameter contains a search condition, the search condition is true for the table space, index space, or partition.
- DSNACCOR recommends at least one action for the table space, index space, or partition.

Table 202 shows the columns of a result set row.

*Table 202. Result set row for second DSNACCOR result set*

| Column name | Data type | Description |
|---|---|---|
| DBNAME | CHAR(8) | Name of the database that contains the object. |
| NAME | CHAR(8) | Table space or index space name. |
| PARTITION | INTEGER | Data set number or partition number. |
| OBJECTTYPE | CHAR(2) | DB2 object type:<br>• TS for a table space<br>• IX for an index space |
| OBJECTSTATUS | CHAR(36) | Status of the object:<br>• ORPHANED, if the object is an index space with no corresponding table space, or the object does not exist<br>• If the object is in a restricted state, one of the following values:<br>– TS=*restricted-state*, if OBJECTTYPE is TS<br>– IX=*restricted-state*, if OBJECTTYPE is IX<br>*restricted-state* is one of the status codes that appear in DISPLAY DATABASE output. See Chapter 2 of *DB2 Command Reference* for details. |
| IMAGECOPY | CHAR(3) | COPY recommendation:<br>• If OBJECTTYPE is TS: FUL (full image copy), INC (incremental image copy), or NO<br>• If OBJECTTYPE is IX: YES or NO |

*Table 202. Result set row for second DSNACCOR result set (continued)*

| Column name | Data type | Description |
|---|---|---|
| RUNSTATS | CHAR(3) | RUNSTATS recommendation: YES or NO. |
| EXTENTS | CHAR(3) | Whether the data sets for the object have exceeded *ExtentLimit*: YES or NO. |
| REORG | CHAR(3) | REORG recommendation: YES or NO. |
| INEXCEPTTABLE | CHAR(40) | A string that contains one of the following values:<br>• Text that you specify in the QUERYTYPE column of the exception table.<br>• YES, if you put a row in the exception table for the object that this result set row represents, but you specify NULL in the QUERYTYPE column.<br>• NO, if the exception table exists but does not have a row for the object that this result set row represents.<br>• Null, if the exception table does not exist, or the *ChkLvl* input parameter does not include the value 4. |
| ASSOCIATEDTS | CHAR(8) | If OBJECTTYPE is IX and the *ChkLvl* input parameter includes the value 2, this value is the name of the table space that is associated with the index space. Otherwise this value is null. |
| COPYLASTTIME | TIMESTAMP | Timestamp of the last full image copy on the object. This value is null if COPY was never run, or the last COPY execution was terminated. |
| LOADRLASTTIME | TIMESTAMP | Timestamp of the last LOAD REPLACE on the object. NULL if LOAD REPLACE was never run, or the last LOAD REPLACE execution was terminated. |
| REBUILDLASTTIME | TIMESTAMP | Timestamp of the last REBUILD INDEX on the object. This value is null if REBUILD INDEX was never run, or if the last REBUILD INDEX execution was terminated. |
| CRUPDPGSPCT | INTEGER | If OBJECTTYPE is TS or IX and IMAGECOPY is YES, the ratio of distinct updated pages to preformatted pages, expressed as a percentage. Otherwise, this value is null. |
| CRCPYCHGPCT | INTEGER | If OBJECTTYPE is TS and IMAGECOPY is YES, this value is the ratio of the number INSERTs, UPDATEs, and DELETEs since the last image copy to the total number of rows or LOBs in the table space or partition, expressed as a percentage. If OBJECTTYPE is IX and IMAGECOPY is YES, this value is the ratio of the number INSERTs and DELETEs since the last image copy to the total number of entries in the index space or partition, expressed as a percentage. Otherwise, this value is null. |
| CRDAYSCELSTCPY | INTEGER | If OBJECTTYPE is TS or IX and IMAGECOPY is YES, the number of days since the last image copy. Otherwise, this value is null. |
| CRINDEXSIZE | INTEGER | If OBJECTTYPE is IX and IMAGECOPY is YES, the number of active pages in the index space or partition. Otherwise, this value is null. |
| REORGLASTTIME | TIMESTAMP | Timestamp of the last REORG on the object. This value is null if REORG was never run, or if the last REORG execution was terminated. |
| RRTINSDELUPDPCT | INTEGER | If OBJECTTYPE is TS and REORG is YES, the ratio of the sum of INSERTs, UPDATEs, and DELETEs since the last REORG to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise, this value is null. |

*Table 202. Result set row for second DSNACCOR result set (continued)*

| Column name | Data type | Description |
|---|---|---|
| RRTUNCINSPCT | INTEGER | If OBJECTTYPE is TS and REORG is YES, the ratio of the number of unclustered INSERTs to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise, this value is null. |
| RRTDISORGLOBPCT | INTEGER | If OBJECTTYPE is TS and REORG is YES, the ratio of the number of imperfectly chunked LOBs to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise, this value is null. |
| RRTMASSDELETE | INTEGER | If OBJECTTYPE is TS, REORG is YES, and the table space is a segmented table space or LOB table space, this value is the number of mass deletes since the last REORG or LOAD REPLACE. If OBJECTTYPE is TS, REORG is YES, and the table space is nonsegmented, this value is the number of dropped tables since the last REORG or LOAD REPLACE. Otherwise, this value is null. |
| RRTINDREF | INTEGER | If OBJECTTYPE is TS, REORG is YES, the ratio of the total number of overflow records that were created since the last REORG or LOAD REPLACE to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise, this value is null. |
| RRIINSDELPCT | INTEGER | If OBJECTTYPE is IX and REORG is YES, the ratio of the sum of INSERTs and DELETEs since the last REORG to the total number of index entries in the index space or partition, expressed as a percentage. Otherwise, this value is null. |
| RRIAPPINSPCT | INTEGER | If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index entries that were inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE and had a key value greater than the maximum key value in the index space or partition, to the number of index entries in the index space or partition, expressed as a percentage. Otherwise, this value is null. |
| RRIPSDDELPCT | INTEGER | If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index entries that were pseudo-deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE to the number of index entries in the index space or partition, expressed as a percentage. Otherwise, this value is null. |
| RRIMASSDELETE | INTEGER | If OBJECTTYPE is IX and REORG is YES, the number of mass deletes from the index space or partition since the last REORG, REBUILD, or LOAD REPLACE. Otherwise, this value is null. |
| RRILEAF | INTEGER | If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was far from the location of the original page, to the total number of active pages in the index space or partition, expressed as a percentage. Otherwise, this value is null. |
| RRINUMLEVELS | INTEGER | If OBJECTTYPE is IX and REORG is YES, the number of levels in the index tree that were added or removed since the last REORG, REBUILD INDEX, or LOAD REPLACE. Otherwise, this value is null. |

*Table 202. Result set row for second DSNACCOR result set (continued)*

| Column name | Data type | Description |
|---|---|---|
| STATSLASTTIME | TIMESTAMP | Timestamp of the last RUNSTATS on the object. This value is null if RUNSTATS was never run, or if the last RUNSTATS execution was terminated. |
| SRTINSDELPCT | INTEGER | If OBJECTTYPE is TS and RUNSTATS is YES, the ratio of the number INSERTs, UPDATEs, and DELETEs since the last RUNSTATS on a table space or partition, to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise, this value is null. |
| SRTINSDELABS | INTEGER | If OBJECTTYPE is TS and RUNSTATS is YES, the number INSERTs, UPDATEs, and DELETEs since the last RUNSTATS on a table space or partition. Otherwise, this value is null. |
| SRTMASSDELETE | INTEGER | If OBJECTTYPE is TS and RUNSTATS is YES, the number of mass deletes from the table space or partition since the last REORG or LOAD REPLACE. Otherwise, this value is null. |
| SRIINSDELPCT | INTEGER | If OBJECTTYPE is IX and RUNSTATS is YES, the ratio of the number INSERTs and DELETEs since the last RUNSTATS on the index space or partition, to the total number of index entries in the index space or partition, expressed as a percentage. Otherwise, this value is null. |
| SRIINSDELABS | INTEGER | If OBJECTTYPE is IX and RUNSTATS is YES, the number INSERTs and DELETEs since the last RUNSTATS on the index space or partition. Otherwise, this value is null. |
| SRIMASSDELETE | INTEGER | If OBJECTTYPE is IX and RUNSTATS is YES, the number of mass deletes from the index space or partition since the last REORG, REBUILD INDEX, or LOAD REPLACE. Otherwise, this value is null. |
| TOTALEXTENTS | SMALLINT | If EXTENTS is YES, the number of physical extents in the table space, index space, or partition. Otherwise, this value is null. |

## The CICS transaction invocation stored procedure (DSNACICS)

The CICS transaction invocation stored procedure (DSNACICS) invokes CICS server programs. DSNACICS gives workstation applications a way to invoke CICS server programs while using TCP/IP as their communication protocol. The workstation applications use TCP/IP and DB2 Connect to connect to a DB2 for OS/390 and z/OS subsystem, and then call DSNACICS to invoke the CICS server programs.

The DSNACICS input parameters require knowledge of various CICS resource definitions with which the workstation programmer might not be familiar. For this reason, DSNACICS invokes the DSNACICX user exit. The system programmer can write a version of DSNACICX that checks and overrides the parameters that the DSNACICS caller passes. If no user version of DSNACICX is provided, DSNACICS invokes the default version of DSNACICX, which does not modify any parameters.

### Environment

DSNACICS runs in a WLM-established stored procedure address space and uses the Recoverable Resource Manager Services attachment facility to connect to DB2.

If you use CICS Transaction Server for OS/390 Version 1 Release 3 or later, you can register your CICS system as a resource manager with recoverable resource management services (RRMS). When you do that, changes to DB2 databases that are made by the program that calls DSNACICS and the CICS server program that DSNACICS invokes are in the same two-phase commit scope. This means that when the calling program performs an SQL COMMIT or ROLLBACK, DB2 and RRS inform CICS about the COMMIT or ROLLBACK.

If the CICS server program that DSNACICS invokes accesses DB2 resources, the server program runs under a separate unit of work from the original unit of work that calls the stored procedure. This means that the CICS server program might deadlock with locks that the client program acquires.

## 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:

- The EXECUTE privilege on stored procedure DSNACICS
- Ownership of the stored procedure
- SYSADM authority

The CICS server program that DSNACICS calls runs under the same user ID as DSNACICS. That user ID depends on the SECURITY parameter that you specify when you define DSNACICS. See Part 2 of *DB2 Installation Guide*.

The DSNACICS caller also needs authorization from an external security system, such as RACF, to use CICS resources. See Part 2 of *DB2 Installation Guide*.

## DSNACICS syntax diagram

The following syntax diagram shows the SQL CALL statement for invoking DSNACICS. Because the linkage convention for DSNACICS is GENERAL WITH NULLS, if you pass parameters in host variables, you need to include a null indicator with every host variable. Null indicators for input host variables must be initialized before you execute the CALL statement.

```
►►──CALL──DSNACICS──(──┬─parm-level─┬──,──┬─pgm-name─┬──,──┬─CICS-applid─┬──,──┬─CICS-level─┬──,──────────►
                       └─NULL───────┘     └─NULL─────┘     └─NULL────────┘     └─NULL───────┘

►──┬─connect-type─┬──,──┬─netname─┬──,──┬─mirror-trans─┬──,──┬─COMMAREA─┬──,──┬─COMMAREA-total-len─┬──,──────►
   └─NULL─────────┘     └─NULL────┘     └─NULL─────────┘     └─NULL─────┘     └─NULL───────────────┘

►──┬─sync-opts─┬──,──return-code,──msg-area──)────────────────────────────────────────────────────────►◄
   └─NULL──────┘
```

## DSNACICS option descriptions

*parm-level*
: Specifies the level of the parameter list that is supplied to the stored procedure. This is an input parameter of type INTEGER. The value must be 1.

*pgm-name*
: Specifies the name of the CICS program that DSNACICS invokes. This is the

name of the program that the CICS mirror transaction calls, *not* the CICS transaction name. This is an input parameter of type CHAR(8).

*CICS-applid*

Specifies the applid of the CICS system to which DSNACICS connects. This is an input parameter of type CHAR(8).

*CICS-level*

Specifies the level of the target CICS subsystem:

**1** The CICS subsystem is CICS for MVS/ESA Version 4 Release 1, CICS Transaction Server for OS/390 Version 1 Release 1, or CICS Transaction Server for OS/390 Version 1 Release 2.

**2** The CICS subsystem is CICS Transaction Server for OS/390 Version 1 Release 3 or later.

This is an input parameter of type INTEGER.

*connect-type*

Specifies whether the CICS connection is generic or specific. Possible values are GENERIC or SPECIFIC. This is an input parameter of type CHAR(8).

*netname*

If the value of *connection-type* is SPECIFIC, specifies the name of the specific connection that is to be used. This value is ignored if the value of *connection-type* is GENERIC. This is an input parameter of type CHAR(8).

*mirror-trans*

Specifies the name of the CICS mirror transaction to invoke. This mirror transaction calls the CICS server program that is specified in the *pgm-name* parameter. *mirror-trans* must be defined to the CICS server region, and the CICS resource definition for *mirror-trans* must specify DFHMIRS as the program that is associated with the transaction.

If this parameter contains blanks, DSNACICS passes a mirror transaction parameter value of null to the CICS EXCI interface. This allows an installation to override the transaction name in various CICS user-replaceable modules. If a CICS user exit does not specify a value for the mirror transaction name, CICS invokes CICS-supplied default mirror transaction CSMI.

This is an input parameter of type CHAR(4).

*COMMAREA*

Specifies the communication area (COMMAREA) that is used to pass data between the DSNACICS caller and the CICS server program that DSNACICS calls. This is an input/output parameter of type VARCHAR(32704). In the length field of this parameter, specify the number of bytes that DSNACICS sends to the CICS server program.

*commarea-total-len*

Specifies the total length of the COMMAREA that the server program needs. This is an input parameter of type INTEGER. This length must be greater than or equal to the value that you specify in the length field of the COMMAREA parameter and less than or equal to 32704. When the CICS server program completes, DSNACICS passes the server program's entire COMMAREA, which is *commarea-total-len* bytes in length, to the stored procedure caller.

*sync-opts*

Specifies whether the calling program controls resource recovery, using two-phase commit protocols that are supported by OS/390 RRS. Possible values are:

**1** The client program controls commit processing. The CICS server region does not perform a syncpoint when the server program returns control to CICS. Also, the server program cannot take any explicit syncpoints. Doing so causes the server program to abnormally terminate.

**2** The target CICS server region takes a syncpoint on successful completion of the server program. If this value is specified, the server program can take explicit syncpoints.

When CICS has been set up to be an RRS resource manager, the client application can control commit processing using SQL COMMIT requests. DB2 for OS/390 and z/OS ensures that CICS is notified to commit any resources that the CICS server program modifies during two-phase commit processing.

When CICS has not been set up to be an RRS resource manager, CICS forces syncpoint processing of all CICS resources at completion of the CICS server program. This commit processing is not coordinated with the commit processing of the client program.

This option is ignored when *CICS-level* is 1. This is an input parameter of type INTEGER.

*return-code*

Return code from the stored procedure. Possible values are:

**0** The call completed successfully.

**12** The request to run the CICS server program failed. The *msg-area* parameter contains messages that describe the error.

This is an output parameter of type INTEGER.

*msg-area*

Contains messages if an error occurs during stored procedure execution. The first messages in this area are generated by the stored procedure. Messages that are generated by CICS or the DSNACICX user exit might follow the first messages. The messages appear as a series of concatenated, viewable text strings. This is an output parameter of type VARCHAR(500).

## DSNACICX user exit

DSNACICS always calls user exit DSNACICX. You can use DSNACICX to change the values of DSNACICS input parameters before you pass those parameters to CICS. If you do not supply your own version of DSNACICX, DSNACICS calls the default DSNACICX, which modifies no values and does an immediate return to DSNACICS. The source code for the default version of DSNACICX is in member DSNASCIX in data set *prefix*.SDSNSAMP. The source code for a sample version of DSNACICX that is written in COBOL is in member DSNASCIO in data set *prefix*.SDSNSAMP.

### General considerations

The DSNACICX exit must follow these rules:

- It can be written in assembler, COBOL, PL/I, or C.
- It must follow the Language Environment calling linkage when the caller is an assembler language program.
- The load module for DSNACICX must reside in an authorized program library that is in the STEPLIB concatenation of the stored procedure address space startup procedure.

  You can replace the default DSNACICX in the *prefix*.SDSNLOAD, library, or you can put the DSNACICX load module in a library that is ahead of *prefix*.SDSNLOAD in the STEPLIB concatenation. It is recommended that you put DSNACICX in the *prefix*.SDSNEXIT library. Sample installation job DSNTIJEX contains JCL for assembling and link-editing the sample source code for DSNACICX into *prefix*.SDSNEXIT. You need to modify the JCL for the libraries and the compiler that you are using.

- The load module must be named DSNACICX.
- The exit must save and restore the caller's registers. Only the contents of register 15 can be modified.
- It must be written to be reentrant and link-edited as reentrant.
- It must be written and link-edited to execute as AMODE(31),RMODE(ANY).
- DSNACICX can contain SQL statements. However, if it does, you need to change the DSNACICS procedure definition to reflect the appropriate SQL access level for the types of SQL statements that you use in the user exit.

## Specifying the routine

DSNACICS always calls an exit routine named DSNACICX. DSNACICS calls your DSNACICX exit routine if it finds it before the default DSNACICX exit routine. Otherwise, it calls the default DSNACICX exit routine.

## When the exit is taken

The DSNACICX exit is taken whenever DSNACICS is called. The exit is taken before DSNACICS invokes the CICS server program.

## Loading a new version of the exit

DB2 loads DSNACICX only once, when DSNACICS is first invoked. If you change DSNACICX, you can load the new version by quiescing and then resuming the WLM application environment for the stored procedure address space in which DSNACICS runs:

```
VARY WLM,APPLENV=DSNACICS-applenv-name,QUIESCE
VARY WLM,APPLENV=DSNACICS-applenv-name,RESUME
```

## Parameter list for DSNACICX

At invocation, registers are set as described in Table 203.

*Table 203. Registers at invocation of DSNACICX*

| Register | Contains |
|---|---|
| **1** | Address of pointer to the exit parameter list (XPL). |
| **13** | Address of the register save area. |
| **14** | Return address. |
| **15** | Address of entry point of exit routine. |

Table 204 shows the contents of the DSNACICX exit parameter list, XPL. Member DSNDXPL in data set *prefix*.SDSNMACS contains an assembler language mapping macro for XPL. Sample exit DSNASCIO in data set *prefix*.SDSNSAMP includes a COBOL mapping macro for XPL.

*Table 204. Contents of the XPL exit parameter list*

| Name | Hex offset | Data type | Description | Corresponding DSNACICS parameter |
|---|---|---|---|---|
| XPL_EYEC | 0 | Character, 4 bytes | Eye-catcher: 'XPL ' | |
| XPL_LEN | 4 | Character, 4 bytes | Length of the exit parameter list | |

*Table 204. Contents of the XPL exit parameter list (continued)*

| Name | Hex offset | Data type | Description | Corresponding DSNACICS parameter |
|---|---|---|---|---|
| XPL_LEVEL | 8 | 4-byte integer | Level of the parameter list | *parm-level* |
| XPL_PGMNAME | C | Character, 8 bytes | Name of the CICS server program | *pgm-name* |
| XPL_CICSAPPLID | 14 | Character, 8 bytes | CICS VTAM applid | *CICS-applid* |
| XPL_CICSLEVEL | 1C | 4-byte integer | Level of CICS code | *CICS-level* |
| XPL_CONNECTTYPE | 20 | Character, 8 bytes | Specific or generic connection to CICS | *connect-type* |
| XPL_NETNAME | 28 | Character, 8 bytes | Name of the specific connection to CICS | *netname* |
| XPL_MIRRORTRAN | 30 | Character, 8 bytes | Name of the mirror transaction that invokes the CICS server program | *mirror-trans* |
| XPL_COMMAREAPTR | 38 | Address, 4 bytes | Address of the COMMAREA | [1] |
| XPL_COMMINLEN | 3C | 4–byte integer | Length of the COMMAREA that is passed to the server program | [2] |
| XPL_COMMTOTLEN | 40 | 4–byte integer | Total length of the COMMAREA that is returned to the caller | *commarea-total-len* |
| XPL_SYNCOPTS | 44 | 4–byte integer | Syncpoint control option | *sync-opts* |
| XPL_RETCODE | 48 | 4–byte integer | Return code from the exit routine | *return-code* |
| XPL_MSGLEN | 4C | 4–byte integer | Length of the output message area | *return-code* |
| XPL_MSGAREA | 50 | Character, 256 bytes | Output message area | *msg-area*[3] |

**Note:**

1. The area that this field points to is specified by DSNACICS parameter *COMMAREA*. This area does not include the length bytes.
2. This is the same value that the DSNACICS caller specifies in the length bytes of the *COMMAREA* parameter.
3. Although the total length of *msg-area* is 500 bytes, DSNACICX can use only 256 bytes of that area.

# Example of DSNACICS invocation

The following PL/I example shows the variable declarations and SQL CALL statement for invoking the CICS transaction that is associated with program CICSPGM1.

```
/***********************/
/* DSNACICS PARAMETERS */
/***********************/
DECLARE  PARM_LEVEL  BIN FIXED(31);
DECLARE  PGM_NAME    CHAR(8);
DECLARE  CICS_APPLID CHAR(8);
DECLARE  CICS_LEVEL  BIN FIXED(31);
DECLARE  CONNECT_TYPE CHAR(8);
DECLARE  NETNAME     CHAR(8);
DECLARE  MIRROR_TRANS CHAR(4);
DECLARE  COMMAREA_TOTAL_LEN BIN FIXED(31);
DECLARE  SYNC_OPTS   BIN FIXED(31);
DECLARE  RET_CODE    BIN FIXED(31);
DECLARE  MSG_AREA    CHAR(500) VARYING;
DECLARE 1 COMMAREA  BASED(P1),
          3 COMMAREA_LEN BIN FIXED(15),
          3 COMMAREA_INPUT   CHAR(30),
          3 COMMAREA_OUTPUT  CHAR(100);

/***********************************************/
/* INDICATOR VARIABLES FOR DSNACICS PARAMETERS */
/***********************************************/
DECLARE 1 IND_VARS,
          3 IND_PARM_LEVEL   BIN FIXED(15),
          3 IND_PGM_NAME     BIN FIXED(15),
          3 IND_CICS_APPLID  BIN FIXED(15),
          3 IND_CICS_LEVEL   BIN FIXED(15),
          3 IND_CONNECT_TYPE BIN FIXED(15),
          3 IND_NETNAME      BIN FIXED(15),
          3 IND_MIRROR_TRANS BIN FIXED(15),
          3 IND_COMMAREA     BIN FIXED(15),
          3 IND_COMMAREA_TOTAL_LEN  BIN FIXED(15),
          3 IND_SYNC_OPTS    BIN FIXED(15),
          3 IND_RETCODE      BIN FIXED(15),
          3 IND_MSG_AREA     BIN FIXED(15);

/**************************/
/* LOCAL COPY OF COMMAREA */
/**************************/
DECLARE P1 POINTER;
DECLARE COMMAREA_STG CHAR(130) VARYING;
/**************************************************************/
/* ASSIGN VALUES TO INPUT PARAMETERS PARM_LEVEL, PGM_NAME,    */
/* MIRROR_TRANS, COMMAREA, COMMAREA_TOTAL_LEN, AND SYNC_OPTS. */
/* SET THE OTHER INPUT PARAMETERS TO NULL. THE DSNACICX       */
/* USER EXIT MUST ASSIGN VALUES FOR THOSE PARAMETERS.         */
/**************************************************************/
PARM_LEVEL = 1;
IND_PARM_LEVEL = 0;

PGM_NAME = 'CICSPGM1';
IND_PGM_NAME = 0 ;

MIRROR_TRANS = 'MIRT';
IND_MIRROR_TRANS = 0;

P1 = ADDR(COMMAREA_STG);
COMMAREA_INPUT = 'THIS IS THE INPUT FOR CICSPGM1';
COMMAREA_OUTPUT = ' ';
COMMAREA_LEN = LENGTH(COMMAREA_INPUT);
IND_COMMAREA = 0;

COMMAREA_TOTAL_LEN = COMMAREA_LEN + LENGTH(COMMAREA_OUTPUT);
IND_COMMAREA_TOTAL_LEN = 0;

SYNC_OPTS = 1;
IND_SYNC_OPTS = 0;

IND_CICS_APPLID= -1;
IND_CICS_LEVEL = -1;
IND_CONNECT_TYPE = -1;
IND_NETNAME = -1;
/*****************************************/
/* INITIALIZE OUTPUT PARAMETERS TO NULL. */
/*****************************************/
IND_RETCODE = -1;
IND_MSG_AREA= -1;
/*****************************************/
/* CALL DSNACICS TO INVOKE CICSPGM1.     */
/*****************************************/
EXEC SQL
```

```
CALL SYSPROC.DSNACICS(:PARM_LEVEL         :IND_PARM_LEVEL,
                      :PGM_NAME           :IND_PGM_NAME,
                      :CICS_APPLID        :IND_CICS_APPLID,
                      :CICS_LEVEL         :IND_CICS_LEVEL,
                      :CONNECT_TYPE       :IND_CONNECT_TYPE,
                      :NETNAME            :IND_NETNAME,
                      :MIRROR_TRANS       :IND_MIRROR_TRANS,
                      :COMMAREA_STG       :IND_COMMAREA,
                      :COMMAREA_TOTAL_LEN :IND_COMMAREA_TOTAL_LEN,
                      :SYNC_OPTS          :IND_SYNC_OPTS,
                      :RET_CODE           :IND_RETCODE,
                      :MSG_AREA           :IND_MSG_AREA);
```

## DSNACICS output

DSNACICS places the return code from DSNACICS execution in the *return-code* parameter. If the value of the return code is non-zero, DSNACICS puts its own error messages and any error messages that are generated by CICS and the DSNACICX user exit in the *msg-area* parameter.

The *COMMAREA* parameter contains the COMMAREA for the CICS server program that DSNACICS calls. The *COMMAREA* parameter has a VARCHAR type. Therefore, if the server program puts data other than character data in the COMMAREA, that data can become corrupted by code page translation as it is passed to the caller. To avoid code page translation, you can change the COMMAREA parameter in the CREATE PROCEDURE statement for DSNACICS to VARCHAR(32704) FOR BIT DATA. However, if you do so, the client program might need to do code page translation on any character data in the COMMAREA to make it readable.

## DSNACICS restrictions

Because DSNACICS uses the distributed program link (DPL) function to invoke CICS server programs, server programs that you invoke through DSNACICS can contain only the CICS API commands that the DPL function supports. The list of supported commands is documented in CICS for MVS/ESA Application Programming Reference.

## DSNACICS debugging

If you receive errors when you call DSNACICS, ask your system administrator to add a DSNDUMP DD statement in the startup procedure for the address space in which DSNACICS runs. The DSNDUMP DD statement causes DB2 to generate an SVC dump whenever DSNACICS issues an error message.

# Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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

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

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

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

## Programming Interface Information

This book is intended to help you to plan for and administer IBM DATABASE 2 Universal Database Server for OS/390 and z/OS (DB2 for OS/390 and z/OS).

This book also documents General-use Programming Interface and Associated Guidance Information and Product-sensitive Programming Interface and Associated Guidance Information provided by IBM DATABASE 2 Universal Database Server for OS/390 and z/OS.

General-use Programming Interfaces allow the customer to write programs that obtain the services of DB2 for OS/390 and z/OS.

General-use Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

**General-use Programming Interface**

General-use Programming Interface and Associated Guidance Information ...

**End of General-use Programming Interface**

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, either by an introductory statement to a chapter or section 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 International Business Machines Corporation in the United States, other countries, or both.

AD/Cycle
APL2
AS/400
BookManager
C/370
CICS
CICS/ESA
CICS/MVS
DATABASE 2
DataHub
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Universal Database
DFSMSdfp
DFSMSdss
DFSMShsm
DFSMS/MVS
DFSORT
DRDA
Distributed Relational Database Architecture
Enterprise Storage Server
Enterprise System/3090
Enterprise System/9000
ESCON
ES/3090
ES/9000
IBM
IBM Registry
IMS
IMS/ESA
Language Environment
MQSeries
MVS/DFP
MVS/ESA
Net.Data
OpenEdition
Operating System/390
OS/2
OS/390
OS/400
Parallel Sysplex
PR/SM
QMF
RACF
RAMAC
RETAIN
RMF
S/390
SAA
SecureWay
SQL/DS
System/380
System/390
VTAM

NetView is a trademark of Tivoli Systems Inc. in the United States, other countries, or both.

JDBC, Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

# Glossary

The following terms and abbreviations are defined as they are used in the DB2 library.

## A

**abend.** Abnormal end of task.

**abend reason code.** A 4-byte hexadecimal code that uniquely identifies a problem with DB2. A complete list of DB2 abend reason codes and their explanations is contained in *DB2 Messages and Codes*.

**abnormal end of task (abend).** Termination of a task, job, or subsystem because of an error condition that recovery facilities cannot resolve during execution.

**access method services.** The facility that is used to define and reproduce VSAM key-sequenced data sets.

**access path.** The path that is used to locate data that is specified in SQL statements. An access path can be indexed or sequential.

**active log.** The portion of the DB2 log to which log records are written as they are generated. The active log always contains the most recent log records, whereas the archive log holds those records that are older and no longer fit on the active log.

**address space.** A range of virtual storage pages that is identified by a number (ASID) and a collection of segment and page tables that map the virtual pages to real pages of the computer's memory.

**address space connection.** The result of connecting an allied address space to DB2. Each address space that contains a task that is connected to DB2 has exactly one address space connection, even though more than one task control block (TCB) can be present. See also *allied address space* and *task control block*.

**agent.** As used in DB2, the structure that associates all processes that are involved in a DB2 unit of work. An *allied agent* is generally synonymous with an *allied thread*. *System agents* are units of work that process independently of the allied agent, such as prefetch processing, deferred writes, and service tasks.

**alias.** An alternative name that can be used in SQL statements to refer to a table or view in the same or a remote DB2 subsystem.

**allied address space.** An area of storage that is external to DB2 and that is connected to DB2. An allied address space is capable of requesting DB2 services.

**allied thread.** A thread that originates at the local DB2 subsystem and that can access data at a remote DB2 subsystem.

**already verified.** An LU 6.2 security option that allows DB2 to provide the user's verified authorization ID when allocating a conversation. The user is not validated by the partner DB2 subsystem.

**ambiguous cursor.** A database cursor that is not defined with the FOR FETCH ONLY clause or the FOR UPDATE OF clause, is not defined on a read-only result table, is not the target of a WHERE CURRENT clause on an SQL UPDATE or DELETE statement, and is in a plan or package that contains either PREPARE or EXECUTE IMMEDIATE SQL statements.

**APAR.** Authorized program analysis report.

**APAR fix corrective service.** A temporary correction of a DB2 defect. The correction is temporary, because it is usually replaced at a later date by a more permanent correction, such as a program temporary fix (PTF).

**APF.** Authorized program facility.

**API.** Application programming interface.

**APPL.** A VTAM network definition statement that is used to define DB2 to VTAM as an application program that uses SNA LU 6.2 protocols.

**application.** A program or set of programs that performs a task; for example, a payroll application.

**application-directed connection.** A connection that an application manages using the SQL CONNECT statement.

**application plan.** The control structure that is produced during the bind process. DB2 uses the application plan to process SQL statements that it encounters during statement execution.

**application process.** The unit to which resources and locks are allocated. An application process involves the execution of one or more programs.

**application programming interface (API).** A functional interface that is supplied by the operating system or by a separately orderable licensed program that allows an application program that is written in a high-level language to use specific data or functions of the operating system or licensed program.

**application requester.** The component on a remote system that generates DRDA requests for data on behalf of an application. An application requester accesses a DB2 database server using the DRDA application-directed protocol.

**application server.** The target of a request from a remote application. In the DB2 environment, the application server function is provided by the distributed data facility and is used to access DB2 data from remote applications.

**archive log.** The portion of the DB2 log that contains log records that have been copied from the active log.

**ASCII.** An encoding scheme that is used to represent strings in many environments, typically on PCs and workstations. Contrast with *EBCDIC* and *Unicode*.

**attachment facility.** An interface between DB2 and TSO, IMS, CICS, or batch address spaces. An attachment facility allows application programs to access DB2.

**attribute.** A characteristic of an entity. For example, in database design, the phone number of an employee is one of that employee's attributes.

**authorization ID.** A string that can be verified for connection to DB2 and to which a set of privileges is allowed. It can represent an individual, an organizational group, or a function, but DB2 does not determine this representation.

**authorized program analysis report (APAR).** A report of a problem that is caused by a suspected defect in a current release of an IBM licensed program.

**authorized program facility (APF).** A facility that permits the identification of programs that are authorized to use restricted functions.

**auxiliary index.** An index on an auxiliary table in which each index entry refers to a LOB.

**auxiliary table.** A table that stores columns outside the table in which they are defined. Contrast with *base table*.

# B

**backward log recovery.** The fourth and final phase of restart processing during which DB2 scans the log in a backward direction to apply UNDO log records for all aborted changes.

**base table.** (1) A table that is created by the SQL CREATE TABLE statement and that holds persistent data. Contrast with *result table* and *temporary table*.

(2) A table containing a LOB column definition. The actual LOB column data is not stored with the base table. The base table contains a row identifier for each row and an indicator column for each of its LOB columns. Contrast with *auxiliary table*.

**base table space.** A table space that contains base tables.

**basic sequential access method (BSAM).** An access method for storing or retrieving data blocks in a continuous sequence, using either a sequential access or a direct access device.

**before trigger.** A trigger that is defined with the trigger activation time BEFORE.

**binary large object (BLOB).** A sequence of bytes where the size of the value ranges from 0 bytes to 2 GB−1. Such a string does not have an associated CCSID.

**binary string.** A sequence of bytes that is not associated with a CCSID. For example, the BLOB data type is a binary string.

**bind.** The process by which the output from the SQL precompiler is converted to a usable control structure, often called an access plan, application plan, or package. During this process, access paths to the data are selected and some authorization checking is performed. The types of bind are:

**automatic bind**. (More correctly, *automatic rebind*) A process by which SQL statements are bound automatically (without a user issuing a BIND command) when an application process begins execution and the bound application plan or package it requires is not valid.

**dynamic bind**. A process by which SQL statements are bound as they are entered.

**incremental bind**. A process by which SQL statements are bound during the execution of an application process, because they could not be bound during the bind process, and VALIDATE(RUN) was specified.

**static bind**. A process by which SQL statements are bound after they have been precompiled. All static SQL statements are prepared for execution at the same time.

**BLOB.** Binary large object.

**BMP.** Batch Message Processing (IMS).

**bootstrap data set (BSDS).** A VSAM data set that contains name and status information for DB2, as well as RBA range specifications, for all active and archive log data sets. It also contains passwords for the DB2 directory and catalog, and lists of conditional restart and checkpoint records.

**BSAM.** Basic sequential access method.

**BSDS.** Bootstrap data set.

**buffer pool.** Main storage that is reserved to satisfy the buffering requirements for one or more table spaces or indexes.

**built-in function.** A function that DB2 supplies. Contrast with *user-defined function*.

## C

**CAF.** Call attachment facility.

**call attachment facility (CAF).** A DB2 attachment facility for application programs that run in TSO or MVS batch. The CAF is an alternative to the DSN command processor and provides greater control over the execution environment.

**cascade delete.** The way in which DB2 enforces referential constraints when it deletes all descendent rows of a deleted parent row.

**cast function.** A function that is used to convert instances of a (source) data type into instances of a different (target) data type. In general, a cast function has the name of the target data type. It has one single argument whose type is the source data type; its return type is the target data type.

**catalog.** In DB2, a collection of tables that contains descriptions of objects such as tables, views, and indexes.

**catalog table.** Any table in the DB2 catalog.

**CCSID.** Coded character set identifier.

**CDB.** Communications database.

**CEC.** Central electronic complex. See *central processor complex*.

**central electronic complex (CEC).** See *central processor complex*.

**central processor complex (CPC).** A physical collection of hardware (such as an ES/3090) that consists of main storage, one or more central processors, timers, and channels.

**character large object (CLOB).** A sequence of bytes representing single-byte characters or a mixture of single- and double-byte characters where the size of the value can be up to 2 GB−1. In general, character large object values are used whenever a character string might exceed the limits of the VARCHAR type.

**character set.** A defined set of characters.

| **character string.** A sequence of bytes that represent
| bit data, single-byte characters, or a mixture of
| single-byte and multibyte characters.

**check constraint.** See *table check constraint*.

**check integrity.** The condition that exists when each row in a table conforms to the table check constraints that are defined on that table. Maintaining check integrity requires DB2 to enforce table check constraints on operations that add or change data.

**check pending.** A state of a table space or partition that prevents its use by some utilities and some SQL statements because of rows that violate referential constraints, table check constraints, or both.

**checkpoint.** A point at which DB2 records internal status information on the DB2 log; the recovery process uses this information if DB2 abnormally terminates.

**CI.** Control interval.

**CICS.** Represents (in this publication) one of the following products:
**CICS Transaction Server for OS/390**: Customer Information Control System Transaction Server for OS/390
**CICS/ESA**: Customer Information Control System/Enterprise Systems Architecture
**CICS/MVS**: Customer Information Control System/Multiple Virtual Storage

**CICS attachment facility.** A DB2 subcomponent that uses the MVS subsystem interface (SSI) and cross storage linkage to process requests from CICS to DB2 and to coordinate resource commitment.

**CIDF.** Control interval definition field.

**claim.** A notification to DB2 that an object is being accessed. Claims prevent drains from occurring until the claim is released, which usually occurs at a commit point. Contrast with *drain*.

**claim class.** A specific type of object access that can be one of the following:
Cursor stability (CS)
Repeatable read (RR)
Write

**claim count.** A count of the number of agents that are accessing an object.

**class of service.** A VTAM term for a list of routes through a network, arranged in an order of preference for their use.

**clause.** In SQL, a distinct part of a statement, such as a SELECT clause or a WHERE clause.

**client.** See *requester*.

**CLIST.** Command list. A language for performing TSO tasks.

**CLOB.** Character large object.

**CLPA.** Create link pack area.

**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.

**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 operation that derives its result by using values from one or more rows. Contrast with *scalar function*.

**"come from" checking.** An LU 6.2 security option that defines a list of authorization IDs that are allowed to connect to DB2 from a partner LU.

**command.** A DB2 operator command or a DSN subcommand. A command is distinct from an SQL statement.

**command recognition character (CRC).** A character that permits an MVS console operator or an IMS subsystem user to route DB2 commands to specific DB2 subsystems.

**commit.** The operation that ends a unit of work by releasing locks so that the database changes that are made by that unit of work can be perceived by other processes.

**commit point.** A point in time when data is considered consistent.

**committed phase.** The second phase of the multisite update process that requests all participants to commit the effects of the logical unit of work.

**common service area (CSA).** In MVS, a part of the common area that contains data areas that are addressable by all address spaces.

**communications database (CDB).** A set of tables in the DB2 catalog that are used to establish conversations with remote database management systems.

**comparison operator.** A token (such as =, >, <) that is used to specify a relationship between two values.

**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 ID.** An identifier that is supplied by the attachment facility and that is associated with a specific address space connection.

**consistency token.** A timestamp that is used to generate the version identifier for an application. See also *version*.

**constraint.** A rule that limits the values that can be inserted, deleted, or updated in a table. See *referential constraint*, *table check constraint*, and *uniqueness constraint*.

**control interval (CI).** A fixed-length area or direct access storage in which VSAM stores records and creates distributed free space. Also, in a key-sequenced data set or file, the set of records pointed to by an entry in the sequence-set index record. The control interval is the unit of information that VSAM transmits to or from direct access storage. A control interval always includes an integral number of physical records.

**control interval definition field (CIDF).** In VSAM, a field located in the 4 bytes at the end of each control interval; it describes the free space, if any, in the control interval.

**conversation.** Communication, which is based on LU 6.2 or Advanced Program-to-Program Communication (APPC), between an application and a remote transaction program over an SNA logical unit-to-logical unit (LU-LU) session that allows communication while processing a transaction.

**coordinator.** The system component that coordinates the commit or rollback of a unit of work that includes work that is done on one or more other systems.

**correlated columns.** A relationship between the value of one column and the value of another column.

**correlated subquery.** A subquery (part of a WHERE or HAVING clause) that is applied to a row or group of rows of a table or view that is named in an outer subselect statement.

**correlation ID.** An identifier that is associated with a specific thread. In TSO, it is either an authorization ID or the job name.

**correlation name.** An identifier that designates a table, a view, or individual rows of a table or view within a single SQL statement. It can be defined in any FROM clause or in the first clause of an UPDATE or DELETE statement.

**cost category.** A category into which DB2 places cost estimates for SQL statements at the time the statement is bound. A cost estimate can be placed in either of the following cost categories:

- A: Indicates that DB2 had enough information to make a cost estimate without using default values.
- B: Indicates that some condition exists for which DB2 was forced to use default values for its estimate.

The cost category is externalized in the COST_CATEGORY column of the DSN_STATEMNT_TABLE when a statement is explained.

**CPC.** Central processor complex.

**CRC.** Command recognition character.

**CRCR.** Conditional restart control record. See also *conditional restart*.

**create link pack area (CLPA).** An option used during IPL to initialize the link pack pageable area.

**created temporary table.** A table that holds temporary data and is defined with the SQL statement CREATE GLOBAL TEMPORARY TABLE. Information about created temporary tables is stored in the DB2 catalog, so this kind of table is persistent and can be shared across application processes. Contrast with *declared temporary table*. See also *temporary table*.

**cross-memory linkage.** A method for invoking a program in a different address space. The invocation is synchronous with respect to the caller.

**CS.** Cursor stability.

**CSA.** Common service area.

**CT.** Cursor 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.** A named control structure that an application program uses to point to a row of interest within some set of rows, and to retrieve rows from the set, possibly making updates or deletions.

**cursor stability (CS).** The isolation level that provides maximum concurrency without the ability to read uncommitted data. With cursor stability, a unit of work holds locks only on its uncommitted changes and on the current row of each of its cursors.

**cursor table (CT).** The copy of the skeleton cursor table that is used by an executing application process.

**cycle.** A set of tables that can be ordered so that each table is a descendent of the one before it, and the first table is a descendent of the last table. A self-referencing table is a cycle with a single member.

# D

**DASD.** Direct access storage device.

**database.** A collection of tables, or a collection of table spaces and index spaces.

**database access thread.** A thread that accesses data at the local subsystem on behalf of a remote subsystem.

**database administrator (DBA).** An individual who is responsible for designing, developing, operating, safeguarding, maintaining, and using a database.

**database descriptor (DBD).** An internal representation of a DB2 database definition, which reflects the data definition that is in the DB2 catalog. The objects that are defined in a database descriptor are table spaces, tables, indexes, index spaces, and relationships.

**database management system (DBMS).** A software system that controls the creation, organization, and modification of a database and the access to the data stored within it.

**database request module (DBRM).** A data set member that is created by the DB2 precompiler and that contains information about SQL statements. DBRMs are used in the bind process.

**database server.** The target of a request from a local application or an intermediate database server. In the DB2 environment, the database server function is provided by the distributed data facility to access DB2 data from local applications, or from a remote database server that acts as an intermediate database server.

**DATABASE 2 Interactive (DB2I).** The DB2 facility that provides for the execution of SQL statements, DB2 (operator) commands, programmer commands, and utility invocation.

**data definition name (ddname).** The name of a data definition (DD) statement that corresponds to a data control block containing the same name.

**Data Language/I (DL/I).** The IMS data manipulation language; a common high-level interface between a user application and IMS.

**data sharing.** The ability of two or more DB2 subsystems to directly access and change a single set of data.

**data sharing group.** A collection of one or more DB2 subsystems that directly access and change the same data while maintaining data integrity.

**data sharing member.** A DB2 subsystem that is assigned by XCF services to a data sharing group.

**data space.** A range of up to 2 GB of contiguous virtual storage addresses that a program can directly manipulate. Unlike an address space, a data space can hold only data; it does not contain common areas, system data, or programs.

**data type.** An attribute of columns, literals, host variables, special registers, and the results of functions and expressions.

**date.** A three-part value that designates a day, month, and year.

**date duration.** A decimal integer that represents a number of years, months, and days.

**datetime value.** A value of the data type DATE, TIME, or TIMESTAMP.

**DBA.** Database administrator.

**DBCLOB.** Double-byte character large object.

**DBCS.** Double-byte character set.

**DBD.** Database descriptor.

**DBID.** Database identifier.

**DBMS.** Database management system.

**DBRM.** Database request module.

**DB2 catalog.** Tables that are maintained by DB2 and contain descriptions of DB2 objects, such as tables, views, and indexes.

**DB2 command.** An instruction to the DB2 subsystem allowing a user to start or stop DB2, to display information on current users, to start or stop databases, to display information on the status of databases, and so on.

**DB2 for VSE & VM.** The IBM DB2 relational database management system for the VSE and VM operating systems.

**DB2I.** DATABASE 2 Interactive.

**DB2I Kanji Feature.** The tape that contains the panels and jobs that allow a site to display DB2I panels in Kanji.

**DB2 PM.** DATABASE 2 Performance Monitor.

**DCLGEN.** Declarations generator.

**DDF.** Distributed data facility.

**ddname.** Data definition name.

**deadlock.** Unresolvable contention for the use of a resource such as a table or an index.

**declarations generator (DCLGEN).** A subcomponent of DB2 that generates SQL table declarations and COBOL, C, or PL/I data structure declarations that conform to the table. The declarations are generated from DB2 system catalog information. DCLGEN is also a DSN subcommand.

**declared temporary table.** A table that holds temporary data and is defined with the SQL statement DECLARE GLOBAL TEMPORARY TABLE. Information about declared temporary tables is not stored in the DB2 catalog, so this kind of table is not persistent and can only be used by the application process that issued the DECLARE statement. Contrast with *created temporary table*. See also *temporary table*.

**default value.** A predetermined value, attribute, or option that is assumed when no other is explicitly specified.

**deferred write.** The process of asynchronously writing changed data pages to disk.

**degree of parallelism.** The number of concurrently executed operations that are initiated to process a query.

**delete rule.** The rule that tells DB2 what to do to a dependent row when a parent row is deleted. For each relationship, the rule might be CASCADE, RESTRICT, SET NULL, or NO ACTION.

**dependent.** An object (row, table, or table space) that has at least one parent. The object is also said to be a dependent (row, table, or table space) of its parent. See *parent row*, *parent table*, *parent table space*.

**dependent row.** A row that contains a foreign key that matches the value of a primary key in the parent row.

**dependent table.** A table that is a dependent in at least one referential constraint.

**descendent.** An object that is a dependent of an object or is the dependent of a descendent of an object.

**descendent row.** A row that is dependent on another row, or a row that is a descendent of a dependent row.

**descendent table.** A table that is a dependent of another table, or a table that is a descendent of a dependent table.

**DFHSM.** Data Facility Hierarchical Storage Manager.

**DFP.** Data Facility Product (in MVS).

**dimension.** A data category such as time, products, or markets. The elements of a dimension are referred to as members. Dimensions offer a very concise, intuitive way of organizing and selecting data for retrieval, exploration, and analysis. See also *dimension table*.

**dimension table.** The representation of a dimension in a star schema. Each row in a dimension table represents all of the attributes for a particular member of the dimension. See also *dimension*, *star schema*, and *star join*.

**direct access storage device (DASD).** A device in which access time is independent of the location of the data.

**directory.** The DB2 system database that contains internal objects such as database descriptors and skeleton cursor tables.

**distinct type.** A user-defined data type that is internally represented as an existing type (its source type), but is considered to be a separate and incompatible type for semantic purposes.

**Distributed Computing Environment MVS/ESA™ (DCE MVS/ESA).** A set of technologies that are provided by the Open Software Foundation to implement distributed computing.

**distributed data facility (DDF).** A set of DB2 components through which DB2 communicates with another RDBMS.

**Distributed Relational Database Architecture (DRDA).** A connection protocol for distributed relational database processing that is used by IBM's relational database products. DRDA includes protocols for communication between an application and a remote relational database management system, and for communication between relational database management systems.

**DL/I.** Data Language/I.

**double-byte character large object (DBCLOB).** A sequence of bytes representing double-byte characters where the size of the values can be up to 2 GB. In general, double-byte character large object values are used whenever a double-byte character string might exceed the limits of the VARGRAPHIC type.

**double-byte character set (DBCS).** A set of characters, which are used by national languages such as Japanese and Chinese, that have more symbols than can be represented by a single byte. Each character is 2 bytes in length. Contrast with *single-byte character set* and *multibyte character set*.

**drain.** The act of acquiring a locked resource by quiescing access to that object.

**drain lock.** A lock on a claim class that prevents a claim from occurring.

**DRDA.** Distributed Relational Database Architecture.

**DRDA access.** An open method of accessing distributed data that you can use to can connect to another database server to execute packages that were previously bound at the server location. You use the SQL CONNECT statement or an SQL statement with a three-part name to identify the server. Contrast with *private protocol access*.

**DSN.** (1) The default DB2 subsystem name. (2) The name of the TSO command processor of DB2. (3) The first three characters of DB2 module and macro names.

**duration.** A number that represents an interval of time. See *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

**EA-enabled table space.** A table space or index space that is enabled for extended addressability and that contains individual partitions (or pieces, for LOB table spaces) that are greater than 4 GB.

**EBCDIC.** Extended binary coded decimal interchange code. An encoding scheme that is used to represent character data in the OS/390, MVS, VM, VSE, and OS/400® environments. Contrast with *ASCII* and *Unicode*.

**EDM pool.** A pool of main storage that is used for database descriptors, application plans, authorization cache, application packages, and dynamic statement caching.

**EID.** Event identifier.

**embedded SQL.** SQL statements that are coded within an application program. See *static SQL*.

**enclave.** In Language Environment, an independent collection of routines, one of which is designated as the main routine. An enclave is similar to a program or run unit.

**EOM.** End of memory.

**EOT.** End of task.

**equijoin.** A join operation in which the join-condition has the form *expression = expression*.

**error page range.** A range of pages that are considered to be physically damaged. DB2 does not allow users to access any pages that fall within this range.

**ESDS.** Entry sequenced data set.

**ESMT.** External subsystem module table (in IMS).

**EUR.** IBM European Standards.

**exception table.** A table that holds rows that violate referential constraints or table check constraints that the CHECK DATA utility finds.

**exclusive lock.** A lock that prevents concurrently executing application processes from reading or changing data. Contrast with *share lock*.

**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.

**expression.** An operand or a collection of operators and operands that yields a single value.

**extended recovery facility (XRF).** A facility that minimizes the effect of failures in MVS, VTAM, the host processor, or high-availability applications during sessions between high-availability applications and designated terminals. This facility provides an alternative subsystem to take over sessions from the failing subsystem.

**external function.** A function for which the body is written in a programming language that takes scalar argument values and produces a scalar result for each invocation. Contrast with *sourced function*, *built-in function*, and *SQL function*.

**external routine.** A user-defined function or stored procedure that is based on code that is written in an external programming language.

**External subsystem module table (ESMT).** The name of the external subsystem module table, which specifies which attachment modules must be loaded by IMS.

# F

**fallback.** The process of returning to a previous release of DB2 after attempting or completing migration to a current release.

**field procedure.** A user-written exit routine that is designed to receive a single value and transform (encode or decode) it in any way the user can specify.

**filter factor.** A number between zero and one that estimates the proportion of rows in a table for which a predicate is true.

**fixed-length string.** A character or graphic string whose length is specified and cannot be changed. Contrast with *varying-length string*.

**foreign key.** A column or set of columns in a dependent table of a constraint relationship. The key must have the same number of columns, with the same descriptions, as the primary key of the parent table. Each foreign key value must either match a parent key value in the related parent table or be null.

**forward log recovery.** The third phase of restart processing during which DB2 processes the log in a forward direction to apply all REDO log records.

**free space.** The total amount of unused space in a page; that is, the space that is not used to store records or control information is free space.

**full outer join.** The result of a join operation that includes the matched rows of both tables that are being joined and preserves the unmatched rows of both tables. See also *join*.

**function.** A mapping, embodied as a program (the function body), invocable by means of zero or more input values (arguments), to a single value (the result). See also *column function* and *scalar function*.

Functions can be user-defined, built-in, or generated by DB2. (See *built-in function*, *cast function*, *external function*, *sourced function*, *SQL function*, and *user-defined function*.)

# G

**GB.** Gigabyte (1 073 741 824 bytes).

**GBP.** Group buffer pool.

**generalized trace facility (GTF).** An MVS service program that records significant system events such as I/O interrupts, SVC interrupts, program interrupts, or external interrupts.

**generic resource name.** A name that VTAM uses to represent several application programs that provide the same function in order to handle session distribution and balancing in a Sysplex environment.

**getpage.** An operation in which DB2 accesses a data page.

**GIMSMP.** The load module name for the System Modification Program/Extended, a basic tool for installing, changing, and controlling changes to programming systems.

**graphic string.** A sequence of DBCS characters.

**gross lock.** The *shared*, *update*, or *exclusive* mode locks on a table, partition, or table space.

**group buffer pool (GBP).** A coupling facility cache structure that is used by a data sharing group to cache data and to ensure that the data is consistent for all members.

**GTF.** Generalized trace facility.

# H

**help panel.** A screen of information presenting tutorial text to assist a user at the terminal.

**hiperspace.** A range of up to 2 GB of contiguous virtual storage addresses that a program can use as a buffer. Like a data space, a hiperspace can hold user data; it does not contain common areas or system data. Unlike an address space or a data space, data in a hiperspace is not directly addressable. To manipulate data in a hiperspace, bring the data into the address space in 4-KB blocks.

**home address space.** The area of storage that MVS currently recognizes as *dispatched*.

**host language.** A programming language in which you can embed SQL statements.

**host program.** An application program that is written in a host language and that contains embedded SQL statements.

**host structure.** In an application program, a structure that is referenced by embedded SQL statements.

**host variable.** In an application program, an application variable that is referenced by embedded SQL statements.

**HSM.** Hierarchical storage manager.

# I

**ICF.** Integrated catalog facility.

**IDCAMS.** An IBM program that is used to process access method services commands. It can be invoked as a job or jobstep, from a TSO terminal, or from within a user’s application program.

**IDCAMS LISTCAT.** A facility for obtaining information that is contained in the access method services catalog.

**identify.** A request that an attachment service program in an address space that is separate from DB2 issues via the MVS subsystem interface to inform DB2 of its existence and to initiate the process of becoming connected to DB2.

**identity column.** A column that provides a way for DB2 to automatically generate a numeric value for each row. The generated values are unique if cycling is not used. Identity columns are defined with the AS IDENTITY clause. Uniqueness of values can be ensured by defining a single-column unique index using the identity column. A table can have no more than one identity column.

**IFCID.** Instrumentation facility component identifier.

**IFI.** Instrumentation facility interface.

**IFI call.** An invocation of the instrumentation facility interface (IFI) by means of one of its defined functions.

**IFP.** IMS Fast Path.

**image copy.** An exact reproduction of all or part of a table space. DB2 provides utility programs to make full image copies (to copy the entire table space) or incremental image copies (to copy only those pages that have been modified since the last image copy).

**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.

**IMS DB.** Information Management System Database.

**IMS TM.** Information Management System Transaction Manager.

**in-abort.** A status of a unit of recovery. If DB2 fails after a unit of recovery begins to be rolled back, but before the process is completed, DB2 continues to back out the changes during restart.

**in-commit.** A status of a unit of recovery. If DB2 fails after beginning its phase 2 commit processing, it "knows," when restarted, that changes made to data are consistent. Such units of recovery are termed *in-commit*.

**independent.** An object (row, table, or table space) that is neither a parent nor a dependent of another object.

**index.** A set of pointers that are logically ordered by the values of a key. Indexes can provide faster access to data and can enforce uniqueness on the rows in a table.

**index key.** The set of columns in a table that is used to determine the order of index entries.

**index partition.** A VSAM data set that is contained within a partitioning index space.

**index space.** A page set that is used to store the entries of one index.

**indicator variable.** A variable that is used to represent the null value in an application program. If the value for the selected column is null, a negative value is placed in the indicator variable.

**indoubt.** A status of a unit of recovery. If DB2 fails after it has finished its phase 1 commit processing and before it has started phase 2, only the commit coordinator knows if an individual unit of recovery is to be committed or rolled back. At emergency restart, if DB2 lacks the information it needs to make this decision, the status of the unit of recovery is *indoubt* until DB2 obtains this information from the coordinator. More than one unit of recovery can be indoubt at restart.

**indoubt resolution.** The process of resolving the status of an indoubt logical unit of work to either the committed or the rollback state.

**inflight.** A status of a unit of recovery. If DB2 fails before its unit of recovery completes phase 1 of the commit process, it merely backs out the updates of its unit of recovery at restart. These units of recovery are termed *inflight*.

**inner join.** The result of a join operation that includes only the matched rows of both tables being joined. See also *join*.

**inoperative package.** A package that cannot be used because one or more user-defined functions or procedures that the package depends on were dropped. Such a package must be explicitly rebound. Contrast with *invalid package.*

**install.** The process of preparing a DB2 subsystem to operate as an MVS subsystem.

**installation verification scenario.** A sequence of operations that exercises the main DB2 functions and tests whether DB2 was correctly installed.

**instrumentation facility component identifier (IFCID).** A value that names and identifies a trace record of an event that can be traced. As a parameter on the START TRACE and MODIFY TRACE commands, it specifies that the corresponding event is to be traced.

**instrumentation facility interface (IFI).** A programming interface that enables programs to obtain online trace data about DB2, to submit DB2 commands, and to pass data to DB2.

**Interactive System Productivity Facility (ISPF).** An IBM licensed program that provides interactive dialog services.

**intermediate database server.** The target of a request from a local application or a remote application requester that is forwarded to another database server. In the DB2 environment, the remote request is forwarded transparently to another database server if the object that is referenced by a three-part name does not reference the local location.

**internal resource lock manager (IRLM).** An MVS subsystem that DB2 uses to control communication and database locking.

**invalid package.** A package that depends on an object (other than a user-defined function) that is dropped. Such a package is implicitly rebound on invocation. Contrast with *inoperative package.*

**IRLM.** Internal resource lock manager.

**ISO.** International Standards Organization.

**isolation level.** The degree to which a unit of work is isolated from the updating operations of other units of work. See also *cursor stability*, *read stability*, *repeatable read*, and *uncommitted read*.

**ISPF.** Interactive System Productivity Facility.

**ISPF/PDF.** Interactive System Productivity Facility/Program Development Facility.

# J

**Japanese Industrial Standards Committee (JISC).** An organization that issues standards for coding character sets.

**Java® Archive (JAR).** A file format that is used for aggregating many files into a single file.

**JCL.** Job control language.

**JES.** MVS Job Entry Subsystem.

**JIS.** Japanese Industrial Standard.

**job control language (JCL).** A control language that is used to identify a job to an operating system and to describe the job’s requirements.

**Job Entry Subsystem (JES).** An IBM licensed program that receives jobs into the system and processes all output data that is produced by the jobs.

**join.** A relational operation that allows retrieval of data from two or more tables based on matching column values. See also *equijoin, full outer join, inner join, left outer join, outer join, and right outer join*.

# K

**KB.** Kilobyte (1024 bytes).

**Kerberos.** A network authentication protocol that is designed to provide strong authentication for client/server applications by using secret-key cryptography.

**Kerberos ticket.** A transparent application mechanism that transmits the identity of an initiating principal to its target. A simple ticket contains the principal’s identity, a session key, a timestamp, and other information, which is sealed using the target’s secret key.

**key.** A column or an ordered collection of columns identified in the description of a table, index, or referential constraint.

**key-sequenced data set (KSDS).** A VSAM file or data set whose records are loaded in key sequence and controlled by an index.

**KSDS.** Key-sequenced data set.

# L

**labeled duration.** A number that represents a duration of years, months, days, hours, minutes, seconds, or microseconds.

**large object (LOB).** A sequence of bytes representing bit data, single-byte characters, double-byte characters, or a mixture of single- and double-byte characters. A LOB can be up to 2 GB−1 byte in length. See also *BLOB*, *CLOB*, and *DBCLOB*.

**latch.** A DB2 internal mechanism for controlling concurrent events or the use of system resources.

**LCID.** Log control interval definition.

**LDS.** Linear data set.

**leaf page.** A page that contains pairs of keys and RIDs and that points to actual data. Contrast with *nonleaf page*.

**left outer join.** The result of a join operation that includes the matched rows of both tables that are being joined, and that preserves the unmatched rows of the first table. See also *join*.

**linear data set (LDS).** A VSAM data set that contains data but no control information. A linear data set can be accessed as a byte-addressable string in virtual storage.

**linkage editor.** A computer program for creating load modules from one or more object modules or load modules by resolving cross references among the modules and, if necessary, adjusting addresses.

**link-edit.** The action of creating a loadable computer program using a linkage editor.

**L-lock.** Logical lock.

**load module.** A program unit that is suitable for loading into main storage for execution. The output of a linkage editor.

**LOB.** Large object.

**LOB lock.** A lock on a LOB value.

**LOB table space.** A table space that contains all the data for a particular LOB column in the related base table.

**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).

**lock.** A means of controlling concurrent events or access to data. DB2 locking is performed by the IRLM.

**lock duration.** The interval over which a DB2 lock is held.

**lock escalation.** The promotion of a lock from a row, page, or LOB lock to a table space lock because the number of page locks that are concurrently held on a given resource exceeds a preset limit.

**locking.** The process by which the integrity of data is ensured. Locking prevents concurrent users from accessing inconsistent data.

**lock mode.** A representation for the type of access that concurrently running programs can have to a resource that a DB2 lock is holding.

**lock object.** The resource that is controlled by a DB2 lock.

**lock promotion.** The process of changing the size or mode of a DB2 lock to a higher level.

**lock size.** The amount of data controlled by a DB2 lock on table data; the value can be a row, a page, a LOB, a partition, a table, or a table space.

**log.** A collection of records that describe the events that occur during DB2 execution and that indicate their sequence. The information thus recorded is used for recovery in the event of a failure during DB2 execution.

**logical index partition.** The set of all keys that reference the same data partition.

**logical lock (L-lock).** The lock type that transactions use to control intra- and inter-DB2 data concurrency between transactions. Contrast with *physical lock (P-lock)*.

**logical recovery pending (LRECP).** The state in which the data and the index keys that reference the data are inconsistent.

**logical unit.** An access point through which an application program accesses the SNA network in order to communicate with another application program.

**logical unit of work (LUW).** The processing that a program performs between synchronization points.

**logical unit of work identifier (LUWID).** A name that uniquely identifies a thread within a network. This name consists of a fully-qualified LU network name, an LUW instance number, and an LUW sequence number.

**log initialization.** The first phase of restart processing during which DB2 attempts to locate the current end of the log.

**log record sequence number (LRSN).** A number that DB2 generates and associates with each log record.

DB2 also uses the LRSN for page versioning. The LRSNs that a particular DB2 data sharing group generates form a strictly increasing sequence for each DB2 log and a strictly increasing sequence for each page across the DB2 group.

**log truncation.** A process by which an explicit starting RBA is established. This RBA is the point at which the next byte of log data is to be written.

**long string.** A string whose actual length, or a varying-length string whose maximum length, is greater than 255 bytes or 127 double-byte characters. Any LOB column, LOB host variable, or expression that evaluates to a LOB is considered a long string.

**LRECP.** Logical recovery pending.

**LRH.** Log record header.

**LRSN.** Log record sequence number.

**LUW.** Logical unit of work.

**LUWID.** Logical unit of work identifier.

# M

**materialize.** (1) The process of putting rows from a view or nested table expression into a work file for additional processing by a query.

(2) The placement of a LOB value into contiguous storage. Because LOB values can be very large, DB2 avoids materializing LOB data until doing so becomes absolutely necessary.

**MB.** Megabyte (1 048 576 bytes).

**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.

**MLPA.** Modified link pack area.

**MODEENT.** A VTAM macro instruction that associates a logon mode name with a set of parameters representing session protocols. A set of MODEENT macro instructions defines a logon mode table.

**mode name.** A VTAM name for the collection of physical and logical characteristics and attributes of a session.

**MPP.** Message processing program (in IMS).

**MSS.** Mass Storage Subsystem.

**MTO.** Master terminal operator.

**multibyte character set (MBCS).** A character set that represents single characters with more than a single byte. Contrast with *single-byte character set* and *double-byte character set*. See also *Unicode*.

**multisite update.** Distributed relational database processing in which data is updated in more than one location within a single unit of work.

**must-complete.** A state during DB2 processing in which the entire operation must be completed to maintain data integrity.

**MVS.** Multiple Virtual Storage.

**MVS/ESA.** Multiple Virtual Storage/Enterprise Systems Architecture.

# N

**nested table expression.** A fullselect in a FROM clause (surrounded by parentheses).

**network identifier (NID).** The network ID that is assigned by IMS or CICS, or if the connection type is RRSAF, the OS/390 RRS unit of recovery ID (URID).

**NID.** Network ID.

**nonleaf page.** A page that contains keys and page numbers of other pages in the index (either leaf or nonleaf pages). Nonleaf pages never point to actual data.

**nonpartitioning index.** Any index that is not a partitioning index.

**NRE.** Network recovery element.

**NUL.** In C, a single character that denotes the end of the string.

**null.** A special value that indicates the absence of information.

**NUL-terminated host variable.** A varying-length host variable in which the end of the data is indicated by the presence of a NUL terminator.

**NUL terminator.** In C, the value that indicates the end of a string. For character strings, the NUL terminator is X'00'.

# O

**OASN (origin application schedule number).** In IMS, a 4-byte number that is assigned sequentially to each IMS schedule since the last cold start of IMS. The OASN is used as an identifier for a unit of work. In an 8-byte format, the first 4 bytes contain the schedule number and the last 4 bytes contain the number of IMS sync points (*commit points*) during the current schedule. The OASN is part of the NID for an IMS connection.

**OBID.** Data object identifier.

**originating task.** In a parallel group, the primary agent that receives data from other execution units (referred to as *parallel tasks*) that are executing portions of the query in parallel.

**OS/390.** Operating System/390.

**OS/390 OpenEdition Distributed Computing Environment (OS/390 OE DCE).** A set of technologies that are provided by the Open Software Foundation to implement distributed computing.

**outer join.** The result of a join operation that includes the matched rows of both tables that are being joined and preserves some or all of the unmatched rows of the tables that are being joined. See also *join*.

# P

**package.** An object containing a set of SQL statements that have been statically bound and that is available for processing. A package is sometimes also called an *application package*.

**package list.** An ordered list of package names that may be used to extend an application plan.

**package name.** The name of an object that is created by a BIND PACKAGE or REBIND PACKAGE command. The object is a bound version of a database request module (DBRM). The name consists of a location name, a collection ID, a package ID, and a version ID.

**page.** A unit of storage within a table space (4 KB, 8 KB, 16 KB, or 32 KB) or index space (4 KB). In a table space, a page contains one or more rows of a table. In a LOB table space, a LOB value can span more than one page, but no more than one LOB value is stored on a page.

**page set.** Another way to refer to a table space or index space. Each page set consists of a collection of VSAM data sets.

**parallel group.** A set of consecutive operations that executed in parallel and that have the same number of parallel tasks.

**parallel I/O processing.** A form of I/O processing in which DB2 initiates multiple concurrent requests for a single user query and performs I/O processing concurrently (in *parallel*) on multiple data partitions.

**Parallel Sysplex.** A set of MVS systems that communicate and cooperate with each other through certain multisystem hardware components and software services to process customer workloads.

**parallel task.** The execution unit that is dynamically created to process a query in parallel. It is implemented by an MVS service request block.

**parent row.** A row whose primary key value is the foreign key value of a dependent row.

**parent table.** A table whose primary key is referenced by the foreign key of a dependent table.

**parent table space.** A table space that contains a parent table. A table space containing a dependent of that table is a dependent table space.

**participant.** An entity other than the commit coordinator that takes part in the commit process. The term participant is synonymous with *agent* in SNA.

**partition.** A portion of a page set. Each partition corresponds to a single, independently extendable data set. Partitions can be extended to a maximum size of 1, 2, or 4 GB, depending on the number of partitions in the partitioned page set. All partitions of a given page set have the same maximum size.

**partitioned data set (PDS).** A data set in direct access storage that is divided into partitions, which are called members. Each partition can contain a program, part of a program, or data. The term partitioned data set is synonymous with program library.

**partitioned page set.** A partitioned table space or an index space. Header pages, space map pages, data pages, and index pages reference data only within the scope of the partition.

**partitioned table space.** A table space that is subdivided into parts (based on index key range), each of which can be processed independently by utilities.

**partner logical unit.** An access point in the SNA network that is connected to the local DB2 subsystem by way of a VTAM conversation.

**PCT.** Program control table (in CICS).

**PDS.** Partitioned data set.

**piece.** A data set of a nonpartitioned page set.

**physical consistency.** The state of a page that is not in a partially changed state.

**plan.** See *application plan*.

**plan allocation.** The process of allocating DB2 resources to a plan in preparation for execution.

**plan name.** The name of an application plan.

**plan segmentation.** The dividing of each plan into sections. When a section is needed, it is independently brought into the EDM pool.

**PLT.** Program list table (in CICS).

**point of consistency.** A time when all recoverable data that an application accesses is consistent with other data. The term point of consistency is synonymous with *sync point* or *commit point*.

**postponed abort UR.** A unit of recovery that was inflight or in-abort, was interrupted by system failure or cancellation, and did not complete backout during restart.

**PPT.** (1) Processing program table (in CICS). (2) Program properties table (in MVS).

**precompilation.** A processing of application programs containing SQL statements that takes place before compilation. SQL statements are replaced with statements that are recognized by the host language compiler. Output from this precompilation includes source code that can be submitted to the compiler and the database request module (DBRM) that is input to the bind process.

**predicate.** An element of a search condition that expresses or implies a comparison operation.

**prefix.** A code at the beginning of a message or record.

**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.

**primary key.** In a relational database, a unique, nonnull key that is part of the definition of a table. A table cannot be defined as a parent unless it has a unique key or primary key.

**principal.** An entity that can communicate securely with another entity. In Kerberos, principals are represented as entries in the Kerberos registry database and include users, servers, computers, and others.

**principal name.** The name by which a principal is known to the DCE security services.

**private connection.** A communications connection that is specific to DB2.

**private protocol access.** A method of accessing distributed data by which you can direct a query to another DB2 system. Contrast with *DRDA access*.

**private protocol connection.** A DB2 private connection of the application process. See also *private connection*.

**privilege.** The capability of performing a specific function, sometimes on a specific object. The term includes:

> **explicit privileges**, which have names and are held as the result of SQL GRANT and REVOKE statements. For example, the SELECT privilege.
>
> **implicit privileges**, which accompany the ownership of an object, such as the privilege to drop a synonym one owns, or the holding of an authority, such as the privilege of SYSADM authority to terminate any utility job.

**privilege set.** For the installation SYSADM ID, the set of all possible privileges. For any other authorization ID, the set of all privileges that are recorded for that ID in the DB2 catalog.

**process.** In DB2, the unit to which DB2 allocates resources and locks. Sometimes called an *application process*, a process involves the execution of one or more programs. The execution of an SQL statement is always associated with some process. The means of initiating and terminating a process are dependent on the environment.

**program.** A single compilable collection of executable statements in a programming language.

**program temporary fix (PTF).** A solution or bypass of a problem that is diagnosed as a result of a defect in a current unaltered release of a licensed program. An authorized program analysis report (APAR) fix is corrective service for an existing problem. A PTF is preventive service for problems that might be encountered by other users of the product. A PTF is *temporary*, because a permanent fix is usually not incorporated into the product until its next release.

**protected conversation.** A VTAM conversation that supports two-phase commit flows.

**PTF.** Program temporary fix.

# Q

**QMF.** Query Management Facility.

**QSAM.** Queued sequential access method.

**query block.** The part of a query that is represented by one of the FROM clauses. Each FROM clause can have multiple query blocks, depending on DB2's internal processing of the query.

**query CP parallelism.** Parallel execution of a single query, which is accomplished by using multiple tasks. See also *Sysplex query parallelism*.

**query I/O parallelism.** Parallel access of data, which is accomplished by triggering multiple I/O requests within a single query.

**queued sequential access method (QSAM).** An extended version of the basic sequential access method (BSAM). When this method is used, a queue of data blocks is formed. Input data blocks await processing, and output data blocks await transfer to auxiliary storage or to an output device.

# R

**RACF.** Resource Access Control Facility, which is a component of the SecureWay Security Server for OS/390.

**RAMAC.** IBM family of enterprise disk storage system products.

**RBA.** Relative byte address.

**RCT.** Resource control table (in CICS attachment facility).

**RDB.** Relational database.

**RDBMS.** Relational database management system.

**RDBNAM.** Relational database name.

**RDF.** Record definition field.

**read stability (RS).** An isolation level that is similar to repeatable read but does not completely isolate an application process from all other concurrently executing application processes. Under level RS, an application that issues the same query more than once might read additional rows that were inserted and committed by a concurrently executing application process.

**rebind.** The creation of a new application plan for an application program that has been bound previously. If, for example, you have added an index for a table that your application accesses, you must rebind the application in order to take advantage of that index.

**record.** The storage representation of a row or other data.

**record identifier (RID).** A unique identifier that DB2 uses internally to identify a row of data in a table stored as a record. Compare with *row ID*.

**record identifier (RID) pool.** An area of main storage above the 16-MB line that is reserved for sorting record identifiers during list prefetch processing.

**recovery.** The process of rebuilding databases after a system failure.

**recovery log.** A collection of records that describes the events that occur during DB2 execution and indicates their sequence. The recorded information is used for recovery in the event of a failure during DB2 execution.

**recovery pending (RECP).** A condition that prevents SQL access to a table space that needs to be recovered.

**recovery token.** An identifier for an element that is used in recovery (for example, *NID* or *URID*).

**RECP.** Recovery pending.

**redo.** A state of a unit of recovery that indicates that changes are to be reapplied to the 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 state of a database in which all values of all foreign keys are valid. Maintaining referential integrity requires the enforcement of referential constraints on all operations that change the data in a table upon which the referential constraints are defined.

**referential structure.** A set of tables and relationships that includes at least one table and, for every table in the set, all the relationships in which that table participates and all the tables to which it is related.

**registry.** See *registry database*.

**registry database.** A database of security information about principals, groups, organizations, accounts, and security policies.

**relational database (RDB).** A database that can be perceived as a set of tables and manipulated in accordance with the relational model of data.

**relational database management system (RDBMS).** A collection of hardware and software that organizes and provides access to a relational database.

**relational database name (RDBNAM).** A unique identifier for an RDBMS within a network. In DB2, this must be the value in the LOCATION column of table SYSIBM.LOCATIONS in the CDB. DB2 publications refer to the name of another RDBMS as a LOCATION value or a location name.

**relationship.** A defined connection between the rows of a table or the rows of two tables. A relationship is the internal representation of a referential constraint.

**relative byte address (RBA).** The offset of a data record or control interval from the beginning of the storage space that is allocated to the data set or file to which it belongs.

**remigration.** The process of returning to a current release of DB2 following a fallback to a previous release. This procedure constitutes another migration process.

**remote attach request.** A request by a remote location to attach to the local DB2 subsystem. Specifically, the request that is sent is an SNA Function Management Header 5.

**remote subsystem.** Any RDBMS, except the *local subsystem*, with which the user or application can communicate. The subsystem need not be remote in any physical sense, and might even operate on the same processor under the same MVS system.

**reoptimization.** The DB2 process of reconsidering the access path of an SQL statement at run time; during reoptimization, DB2 uses the values of host variables, parameter markers, or special registers.

**REORG pending (REORP).** A condition that restricts SQL access and most utility access to an object that must be reorganized.

**REORP.** REORG pending.

**repeatable read (RR).** The isolation level that provides maximum protection from other executing application programs. When an application program executes with repeatable read protection, rows referenced by the program cannot be changed by other programs until the program reaches a commit point.

**request commit.** The vote that is submitted to the prepare phase if the participant has modified data and is prepared to commit or roll back.

**requester.** The source of a request to access data at a remote server. In the DB2 environment, the requester function is provided by the distributed data facility.

**resource allocation.** The part of plan allocation that deals specifically with the database resources.

**resource control table (RCT).** A construct of the CICS attachment facility, created by site-provided macro parameters, that defines authorization and access attributes for transactions or transaction groups.

**resource definition online.** A CICS feature that you use to define CICS resources online without assembling tables.

**resource limit facility (RLF).** A portion of DB2 code that prevents dynamic manipulative SQL statements from exceeding specified time limits. The resource limit facility is sometimes called the governor.

**resource limit specification table.** A site-defined table that specifies the limits to be enforced by the resource limit facility.

**restart pending (RESTP).** A restrictive state of a page set or partition that indicates that restart (backout) work needs to be performed on the object. All access to the page set or partition is denied except for access by the:

- RECOVER POSTPONED command
- Automatic online backout (which DB2 invokes after restart if the system parameter LBACKOUT=AUTO)

**RESTP.** Restart pending.

**result table.** The set of rows that are specified by a SELECT statement.

**RID.** Record identifier.

**RID pool.** Record identifier pool.

**right outer join.** The result of a join operation that includes the matched rows of both tables that are being joined and preserves the unmatched rows of the second join operand. See also *join*.

**RLF.** Resource limit facility.

**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*.

**root page.** The page of an index page set that follows the first index space map page. A root page is the highest level (or the beginning point) of the index.

**routine.** A term that refers to either a user-defined function or a stored procedure.

**row.** The horizontal component of a table. A row consists of a sequence of values, one for each column of the table.

**ROWID.** Row identifier.

**row identifier (ROWID).** A value that uniquely identifies a row. This value is stored with the row and never changes.

**row lock.** A lock on a single row of data.

**RRE.** Residual recovery entry (in IMS).

**RRSAF.** Recoverable Resource Manager Services attachment facility. RRSAF is a DB2 subcomponent that uses OS/390 Transaction Management and Recoverable Resource Manager Services to coordinate resource commitment between DB2 and all other resource managers that also use OS/390 RRS in an OS/390 system.

**RS.** Read stability.

**RTT.** Resource translation table.

# S

**savepoint.** A named entity that represents the state of data and schemas at a particular point in time within a unit of work. SQL statements exist to set a savepoint, release a savepoint, and restore data and schemas to the state that the savepoint represents. The restoration of data and schemas to a savepoint is usually referred to as *rolling back to a savepoint*.

**SBCS.** Single-byte character set.

**scalar function.** An SQL operation that produces a single value from another value and is expressed as a function name, followed by a list of arguments that are enclosed in parentheses. Contrast with *column function*.

**schema.** A logical grouping for user-defined functions, distinct types, triggers, and stored procedures. When an object of one of these types is created, it is assigned to one schema, which is determined by the name of the object. For example, the following statement creates a distinct type T in schema C:

```
CREATE DISTINCT TYPE C.T ...
```

**SDWA.** System diagnostic work area.

**search condition.** A criterion for selecting rows from a table. A search condition consists of one or more predicates.

**secondary authorization ID.** An authorization ID that has been associated with a primary authorization ID by an authorization exit routine.

**section.** The segment of a plan or package that contains the executable structures for a single SQL statement. For most SQL statements, one section in the plan exists for each SQL statement in the source program. However, for cursor-related statements, the DECLARE, OPEN, FETCH, and CLOSE statements reference the same section because, they each refer to the SELECT statement that is named in the DECLARE CURSOR statement. SQL statements such as COMMIT, ROLLBACK, and some SET statements do not use a section.

**segmented table space.** A table space that is divided into equal-sized groups of pages called segments. Segments are assigned to tables so that rows of different tables are never stored in the same segment.

**self-referencing constraint.** A referential constraint that defines a relationship in which a table is a dependent of itself.

**self-referencing table.** A table with a self-referencing constraint.

**sequential data set.** A non-DB2 data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape. Several of the DB2 database utilities require sequential data sets.

**sequential prefetch.** A mechanism that triggers consecutive asynchronous I/O operations. Pages are fetched before they are required, and several pages are read with a single I/O operation.

**server.** The target of a request from a remote requester. In the DB2 environment, the server function is provided by the distributed data facility, which is used to access DB2 data from remote applications.

**service class.** An eight-character identifier that is used by MVS Workload Manager to associate customer performance goals with a particular DDF thread or stored procedure. A service class is also used to classify work on parallelism assistants.

**session.** A link between two nodes in a VTAM network.

**session protocols.** The available set of SNA communication requests and responses.

**share lock.** A lock that prevents concurrently executing application processes from changing data, but not from reading data. Contrast with *exclusive lock*.

**short string.** A string whose actual length, or a varying-length string whose maximum length, is 255 bytes (or 127 double-byte characters) or less. Regardless of length, a LOB string is not a short string.

**sign-on.** A request that is made on behalf of an individual CICS or IMS application process by an attachment facility to enable DB2 to verify that it is authorized to use DB2 resources.

**simple page set.** A nonpartitioned page set. A simple page set initially consists of a single data set (page set piece). If and when that data set is extended to 2 GB, another data set is created, and so on up to a total of 32 data sets. DB2 considers the data sets to be a single contiguous linear address space containing a maximum of 64 GB. Data is stored in the next available location within this address space without regard to any partitioning scheme.

**simple table space.** A table space that is neither partitioned nor segmented.

**single-byte character set (SBCS).** A set of characters in which each character is represented by a single byte. Contrast with *double-byte character set* or *multibyte character set*.

**SMF.** System management facility.

**SMP/E.** System Modification Program/Extended.

**SMS.** Storage Management Subsystem.

**SNA.** Systems Network Architecture.

**SNA network.** The part of a network that conforms to the formats and protocols of Systems Network Architecture (SNA).

**sourced function.** A function that is implemented by another built-in or user-defined function that is already known to the database manager. This function can be a scalar function or a column (aggregating) function; it returns a single value from a set of values (for example, MAX or AVG). Contrast with *built-in function, external function,* and *SQL function*.

**special register.** A storage area that DB2 defines for an application process to use for storing information that can be referenced in SQL statements. Examples of special registers are USER and CURRENT DATE.

**SPUFI.** SQL Processor Using File Input.

**SQL.** Structured Query Language.

**SQL authorization ID (SQL ID).** The authorization ID that is used for checking dynamic SQL statements in some situations.

**SQLCA.** SQL communication area.

**SQL communication area (SQLCA).** A structure that is used to provide an application program with information about the execution of its SQL statements.

**SQLDA.** SQL descriptor area.

**SQL descriptor area (SQLDA).** A structure that describes input variables, output variables, or the columns of a result table.

**SQL/DS.** Structured Query Language/Data System. This product is now obsolete and has been replaced by DB2 for VSE & VM.

**SQL function.** A user-defined function in which the CREATE FUNCTION statement contains the source code. The source code is a single SQL expression that evaluates to a single value. The SQL user-defined function can return only one parameter.

**SQL processing conversation.** Any conversation that requires access of DB2 data, either through an application or by dynamic query requests.

**SQL Processor Using File Input (SPUFI).** 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 routine.** A user-defined function or stored procedure that is based on code that is written in SQL.

**SSI.** Subsystem interface (in MVS).

**SSM.** Subsystem member.

**stand-alone.** An attribute of a program that means it is capable of executing separately from DB2, without using DB2 services.

**star join.** A method of joining a dimension column of a fact table to the key column of the corresponding dimension table. See also *join*, *dimension*, and *star schema*.

**star schema.** The combination of a fact table (which contains most of the data) and a number of dimension tables. See also *star join*, *dimension*, and *dimension table*.

**statement string.** For a dynamic SQL statement, the character string form of the statement.

**static SQL.** SQL statements, embedded within a program, that are prepared during the program preparation process (before the program is executed). After being prepared, the SQL statement does not change (although values of host variables that are specified by the statement might change).

**storage group.** A named set of disks on which DB2 data can be stored.

**stored procedure.** A user-written application program that can be invoked through the use of the SQL CALL statement.

**string.** See *character string* or *graphic string*.

**Structured Query Language (SQL).** A standardized language for defining and manipulating data in a relational database.

**subcomponent.** A group of closely related DB2 modules that work together to provide a general function.

**subpage.** The unit into which a physical index page can be divided.

**subquery.** A SELECT statement within the WHERE or HAVING clause of another SQL statement; a nested SQL statement.

**subselect.** That form of a query that does not include ORDER BY clause, UPDATE clause, or UNION operators.

**subsystem.** A distinct instance of a relational database management system (RDBMS).

**sync point.** See *commit point*.

**synonym.** In SQL, an alternative name for a table or view. Synonyms can be used only to refer to objects at the subsystem in which the synonym is defined.

**Sysplex.** See *Parallel Sysplex*.

**Sysplex query parallelism.** Parallel execution of a single query that is accomplished by using multiple tasks on more than one DB2 subsystem. See also *query CP parallelism*.

**system administrator.** The person at a computer installation who designs, controls, and manages the use of the computer system.

**system agent.** A work request that DB2 creates internally such as prefetch processing, deferred writes, and service tasks.

**system conversation.** The conversation that two DB2 subsystems must establish to process system messages before any distributed processing can begin.

**system diagnostic work area (SDWA).** The data that is recorded in a SYS1.LOGREC entry that describes a program or hardware error.

**system-directed connection.** A connection that an RDBMS manages by processing SQL statements with three-part names.

**System Modification Program/Extended (SMP/E).** A tool for making software changes in programming systems (such as DB2) and for controlling those changes.

**Systems Network Architecture (SNA).** The description of the logical structure, formats, protocols, and operational sequences for transmitting information through and controlling the configuration and operation of networks.

**SYS1.DUMPxx data set.** A data set that contains a system dump.

**SYS1.LOGREC.** A service aid that contains important information about program and hardware errors.

# T

**table.** A named data object consisting of a specific number of columns and some number of unordered rows. See also *base table* or *temporary table*.

**table check constraint.** A user-defined constraint that specifies the values that specific columns of a base table can contain.

**table function.** A function that receives a set of arguments and returns a table to the SQL statement that references the function. A table function can be referenced only in the FROM clause of a subselect.

**table space.** A page set that is used to store the records in one or more tables.

**table space set.** A set of table spaces and partitions that should be recovered together for one of these reasons:

- Each of them contains a table that is a parent or descendent of a table in one of the others.
- The set contains a base table and associated auxiliary tables.

A table space set can contain both types of relationships.

**task control block (TCB).** A control block that is used to communicate information about tasks within an address space that are connected to DB2. An address space can support many task connections (as many as one per task), but only one address space connection. See also *address space connection*.

**TB.** Terabyte (1 099 511 627 776 bytes).

**TCB.** Task control block (in MVS).

**temporary table.** A table that holds temporary data; for example, temporary tables are useful for holding or sorting intermediate results from queries that contain a large number of rows. The two kinds of temporary table, which are created by different SQL statements, are the created temporary table and the declared temporary table. Contrast with *result table*. See also *created temporary table* and *declared temporary table*.

**thread.** The DB2 structure that describes an application's connection, traces its progress, processes resource functions, and delimits its accessibility to DB2 resources and services. Most DB2 functions execute under a thread structure. See also *allied thread* and *database access thread*.

**three-part name.** The full name of a table, view, or alias. It consists of a location name, authorization ID, and an object name, separated by a period.

**time.** A three-part value that designates a time of day in hours, minutes, and seconds.

**time duration.** A decimal integer that represents a number of hours, minutes, and seconds.

**timeout.** Abnormal termination of either the DB2 subsystem or of an application because of the unavailability of resources. Installation specifications are set to determine both the amount of time DB2 is to wait for IRLM services after starting, and the amount of time IRLM is to wait if a resource that an application requests is unavailable. If either of these time specifications is exceeded, a timeout is declared.

**Time-Sharing Option (TSO).** An option in MVS that provides interactive time sharing from remote terminals.

**timestamp.** A seven-part value that consists of a date and time. The timestamp is expressed in years, months, days, hours, minutes, seconds, and microseconds.

**TMP.** Terminal Monitor Program.

**to-do.** A state of a unit of recovery that indicates that the unit of recovery's changes to recoverable DB2 resources are indoubt and must either be applied to the DASD media or backed out, as determined by the commit coordinator.

**trace.** A DB2 facility that provides the ability to monitor and collect DB2 monitoring, auditing, performance, accounting, statistics, and serviceability (global) data.

**TSO.** Time-Sharing Option.

**TSO attachment facility.** A DB2 facility consisting of the DSN command processor and DB2I. Applications that are not written for the CICS or IMS environments can run under the TSO attachment facility.

**type 1 indexes.** Indexes that were created by a release of DB2 before DB2 Version 4 or that are specified as type 1 indexes in Version 4. Contrast with *type 2 indexes*. As of Version 7, type 1 indexes are no longer supported.

**type 2 indexes.** Indexes that are created on a release of DB2 after Version 6 or that are specified as type 2 indexes in Version 4 or later.

# U

**UDF.** User-defined function.

**UDT.** User-defined data type. In DB2 for OS/390 and z/OS, the term *distinct type* is used instead of user-defined data type. See *distinct type*.

**uncommitted read (UR).** The isolation level that allows an application to read uncommitted data.

**undo.** A state of a unit of recovery that indicates that the changes the unit of recovery made to recoverable DB2 resources must be backed out.

**Unicode.** A standard that parallels the ISO-10646 standard. Several implementations of the Unicode standard exist, all of which have the ability to represent a large percentage of the characters contained in the many scripts that are used throughout the world.

**union.** An SQL operation that combines the results of two select statements. Unions are often used to merge lists of values that are obtained from several tables.

**unique constraint.** An SQL rule that no two values in a primary key, or in the key of a unique index, can be the same.

**unique index.** An index which ensures that no identical key values are stored in a table.

**unlock.** The act of releasing an object or system resource that was previously locked and returning it to general availability within DB2.

**UR.** Uncommitted read.

**URE.** Unit of recovery element.

**URID (unit of recovery ID).** The LOGRBA of the first log record for a unit of recovery. The URID also appears in all subsequent log records for that unit of recovery.

**user-defined data type (UDT).** See *distinct type*.

**user-defined function (UDF).** A function that is defined to DB2 by using the CREATE FUNCTION statement and that can be referenced thereafter in SQL statements. A user-defined function can be an *external function,* a *sourced function,* or an *SQL function.* Contrast with *built-in function*.

**UT.** Utility-only access.

# V

**value.** The smallest unit of data that is manipulated in SQL.

**varying-length string.** A character or graphic string whose length varies within set limits. Contrast with *fixed-length string*.

**version.** A member of a set of similar programs, DBRMs, packages, or LOBs.

**A version of a program** is the source code that is produced by precompiling the program. The program version is identified by the program name and a timestamp (consistency token).

**A version of a DBRM** is the DBRM that is produced by precompiling a program. The DBRM version is identified by the same program name and timestamp as a corresponding program version.

**A version of a package** is the result of binding a DBRM within a particular database system. The package version is identified by the same program name and consistency token as the DBRM.

**A version of a LOB** is a copy of a LOB value at a point in time. The version number for a LOB is stored in the auxiliary index entry for the LOB.

**view.** An alternative representation of data from one or more tables. A view can include all or some of the columns that are contained in tables on which it is defined.

**Virtual Storage Access Method (VSAM).** An access method for direct or sequential processing of fixed- and varying-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by relative-record number.

**Virtual Telecommunications Access Method (VTAM).** An IBM licensed program that controls communication and the flow of data in an SNA network.

**VSAM.** Virtual storage access method.

**VTAM.** Virtual Telecommunication Access Method (in MVS).

## W

**WLM application environment.** An MVS Workload Manager attribute that is associated with one or more stored procedures. The WLM application environment determines the address space in which a given DB2 stored procedure runs.

**write to operator (WTO).** An optional user-coded service that allows a message to be written to the system console operator informing the operator of errors and unusual system conditions that may need to be corrected.

**WTO.** Write to operator.

**WTOR.** Write to operator (WTO) with reply.

## X

**XRF.** Extended recovery facility.

## Z

**z/OS.** An operating system for the eServer product line that supports 64-bit real storage.

# Bibliography

**DB2 Universal Database Server for OS/390 and z/OS Version 7 product libraries:**

***DB2 for OS/390 and z/OS***

- *DB2 Administration Guide, SC26-9931*
- *DB2 Application Programming and SQL Guide, SC26-9933*
- *DB2 Application Programming Guide and Reference for Java, SC26-9932*
- *DB2 Command Reference, SC26-9934*
- *DB2 Data Sharing: Planning and Administration, SC26-9935*
- *DB2 Data Sharing Quick Reference Card, SX26-3846*
- *DB2 Diagnosis Guide and Reference, LY37-3740*
- *DB2 Diagnostic Quick Reference Card, LY37-3741*
- *DB2 Image, Audio, and Video Extenders Administration and Programming, SC26-9947*
- *DB2 Installation Guide, GC26-9936*
- *DB2 Licensed Program Specifications, GC26-9938*
- *DB2 Master Index, SC26-9939*
- *DB2 Messages and Codes, GC26-9940*
- *DB2 ODBC Guide and Reference, SC26-9941*
- *DB2 Reference for Remote DRDA Requesters and Servers, SC26-9942*
- *DB2 Reference Summary, SX26-3847*
- *DB2 Release Planning Guide, SC26-9943*
- *DB2 SQL Reference, SC26-9944*
- *DB2 Text Extender Administration and Programming, SC26-9948*
- *DB2 Utility Guide and Reference, SC26-9945*
- *DB2 What's New? GC26-9946*
- *DB2 XML Extender for OS/390 and z/OS Administration and Programming*, SC27-9949
- *DB2 Program Directory, GI10-8182*

***DB2 Administration Tool***

- *DB2 Administration Tool for OS/390 and z/OS User’s Guide, SC26-9847*

***DB2 Buffer Pool Tool***

- *DB2 Buffer Pool Tool for OS/390 and z/OS User’s Guide and Reference, SC26-9306*

***DB2 DataPropagator™***

- *DB2 UDB Replication Guide and Reference, SC26-9920*

***Net.Data®***

The following books are available at this Web site: http://www.ibm.com/software/net.data/library.html

- *Net.Data Library: Administration and Programming Guide for OS/390 and z/OS*
- *Net.Data Library: Language Environment Interface Reference*
- *Net.Data Library: Messages and Codes*
- *Net.Data Library: Reference*

***DB2 PM for OS/390***

- *DB2 PM for OS/390 Batch User's Guide, SC27-0857*
- *DB2 PM for OS/390 Command Reference, SC27-0855*
- *DB2 PM for OS/390 Data Collector Application Programming Interface Guide, SC27-0861*
- *DB2 PM for OS/390 General Information, GC27-0852*
- *DB2 PM for OS/390 Installation and Customization, SC27-0860*
- *DB2 PM for OS/390 Messages, SC27-0856*
- *DB2 PM for OS/390 Online Monitor User's Guide, SC27-0858*
- *DB2 PM for OS/390 Report Reference Volume 1, SC27-0853*
- *DB2 PM for OS/390 Report Reference Volume 2, SC27-0854*
- *DB2 PM for OS/390 Using the Workstation Online Monitor, SC27-0859*
- *DB2 PM for OS/390 Program Directory, GI10-8223*

***Query Management Facility (QMF)***

- *Query Management Facility: Developing QMF Applications, SC26-9579*
- *Query Management Facility: Getting Started with QMF on Windows, SC26-9582*
- *Query Management Facility: High Peformance Option User’s Guide for OS/390 and z/OS, SC26-9581*
- *Query Management Facility: Installing and Managing QMF on OS/390 and z/OS, GC26-9575*

- *Query Management Facility: Installing and Managing QMF on Windows, GC26-9583*
- *Query Management Facility: Introducing QMF, GC26-9576*
- *Query Management Facility: Messages and Codes, GC26-9580*
- *Query Management Facility: Reference, SC26-9577*
- *Query Management Facility: Using QMF, SC26-9578*

**Ada/370**
- *IBM Ada/370 Language Reference, SC09-1297*
- *IBM Ada/370 Programmer's Guide, SC09-1414*
- *IBM Ada/370 SQL Module Processor for DB2 Database Manager User's Guide, SC09-1450*

**APL2®**
- *APL2 Programming Guide, SH21-1072*
- *APL2 Programming: Language Reference, SH21-1061*
- *APL2 Programming: Using Structured Query Language (SQL), SH21-1057*

**AS/400®**

The following books are available at this Web site: www.as400.ibm.com/infocenter
- *DB2 Universal Database for AS/400 Database Programming*
- *DB2 Universal Database for AS/400 Performance and Query Optimization*
- *DB2 Universal Database for AS/400 Distributed Data Management*
- *DB2 Universal Database for AS/400 Distributed Data Programming*
- *DB2 Universal Database for AS/400 SQL Programming Concepts*
- *DB2 Universal Database for AS/400 SQL Programming with Host Languages*
- *DB2 Universal Database for AS/400 SQL Reference*

**BASIC**
- *IBM BASIC/MVS Language Reference, GC26-4026*
- *IBM BASIC/MVS Programming Guide, SC26-4027*

**BookManager® READ/MVS**
- *BookManager READ/MVS V1R3: Installation Planning & Customization, SC38-2035*

**SAA® AD/Cycle® C/370™**
- *IBM SAA AD/Cycle C/370 Programming Guide, SC09-1841*
- *IBM SAA AD/Cycle C/370 Programming Guide for Language Environment/370, SC09-1840*
- *IBM SAA AD/Cycle C/370 User's Guide, SC09-1763*
- *SAA CPI C Reference, SC09-1308*

**Character Data Representation Architecture**
- *Character Data Representation Architecture Overview, GC09-2207*
- *Character Data Representation Architecture Reference and Registry, SC09-2190*

**CICS/ESA**
- *CICS/ESA Application Programming Guide, SC33-1169*
- *CICS External Interfaces Guide, SC33-1944*
- *CICS for MVS/ESA Application Programming Reference, SC33-1170*
- *CICS for MVS/ESA CICS-RACF Security Guide, SC33-1185*
- *CICS for MVS/ESA CICS-Supplied Transactions, SC33-1168*
- *CICS for MVS/ESA Customization Guide, SC33-1165*
- *CICS for MVS/ESA Data Areas, LY33-6083*
- *CICS for MVS/ESA Installation Guide, SC33-1163*
- *CICS for MVS/ESA Intercommunication Guide, SC33-1181*
- *CICS for MVS/ESA Messages and Codes, GC33-1177*
- *CICS for MVS/ESA Operations and Utilities Guide, SC33-1167*
- *CICS/ESA Performance Guide, SC33-1183*
- *CICS/ESA Problem Determination Guide, SC33-1176*
- *CICS for MVS/ESA Resource Definition Guide, SC33-1166*
- *CICS for MVS/ESA System Definition Guide, SC33-1164*
- *CICS for MVS/ESA System Programming Reference, GC33-1171*

**CICS Transaction Server for OS/390**
- *CICS Application Programming Guide, SC33-1687*
- *CICS External Interfaces Guide, SC33-1703*
- *CICS DB2 Guide, SC33-1939*
- *CICS Resource Definition Guide, SC33-1684*

**IBM C/C++ for MVS/ESA**
- *IBM C/C++ for MVS/ESA Library Reference, SC09-1995*
- *IBM C/C++ for MVS/ESA Programming Guide, SC09-1994*

**IBM COBOL**
- *IBM COBOL Language Reference, SC26-4769*
- *IBM COBOL for MVS & VM Programming Guide, SC26-4767*

*IBM COBOL for OS/390 & VM Programming Guide, SC26-9049*

**Conversion Guide**
- *IMS-DB and DB2 Migration and Coexistence Guide, GH21-1083*

**Cooperative Development Environment**
- *CoOperative Development Environment/370: Debug Tool, SC09-1623*

**DataPropagator NonRelational**
- *DataPropagator NonRelational MVS/ESA Administration Guide, SH19-5036*
- *DataPropagator NonRelational MVS/ESA Reference, SH19-5039*

**Data Facility Data Set Services**
- *Data Facility Data Set Services: User's Guide and Reference, SC26-4388*

**Database Design**
- *DB2 Design and Development Guide* by Gabrielle Wiorkowski and David Kull, Addison Wesley, ISBN 0-20158-049-7
- *Handbook of Relational Database Design* by C. Fleming and B. Von Halle, Addison Wesley, ISBN 0-20111-434-8

**DataHub®**
- *IBM DataHub General Information, GC26-4874*

**Data Refresher**
- *Data Refresher Relational Extract Manager for MVS* GI10-9927

**DB2 Connect**
- *DB2 Connect Enterprise Edition for OS/2 and Windows: Quick Beginnings, GC09-2953*
- *DB2 Connect Enterprise Edition for UNIX: Quick Beginnings, GC09-2952*
- *DB2 Connect Personal Edition Quick Beginnings, GC09-2967*
- *DB2 Connect User's Guide, SC09-2954*

**DB2 Red Books**
- *DB2 UDB Server for OS/390 Version 6 Technical Update, SG24-6108-00*

**DB2 Server for VSE & VM**
- *DB2 Server for VM: DBS Utility, SC09-2394*
- *DB2 Server for VSE: DBS Utility, SC09-2395*

**DB2 Universal Database for UNIX, Windows, OS/2®**
- *DB2 UDB Administration Guide: Planning, SC09-2946*
- *DB2 UDB Administration Guide: Implementation, SC09-2944*
- *DB2 UDB Administration Guide: Performance, SC09-2945*
- *DB2 UDB Administrative API Reference, SC09-2947*
- *DB2 UDB Application Building Guide, SC09-2948*
- *DB2 UDB Application Development Guide, SC09-2949*
- *DB2 UDB CLI Guide and Reference, SC09-2950*
- *DB2 UDB SQL Getting Started, SC09-2973*
- *DB2 UDB SQL Reference Volume 1, SC09-2974*
- *DB2 UDB SQL Reference Volume 2, SC09-2975*

**Device Support Facilities**
- *Device Support Facilities User's Guide and Reference, GC35-0033*

**DFSMS**

These books provide information about a variety of components of DFSMS, including DFSMS/MVS, DFSMSdfp, DFSMSdss, DFSMShsm, and MVS/DFP.
- *DFSMS/MVS: Access Method Services for the Integrated Catalog, SC26-4906*
- *DFSMS/MVS: Access Method Services for VSAM Catalogs, SC26-4905*
- *DFSMS/MVS: Administration Reference for DFSMSdss, SC26-4929*
- *DFSMS/MVS: DFSMShsm Managing Your Own Data, SH21-1077*
- *DFSMS/MVS: Diagnosis Reference for DFSMSdfp, LY27-9606*
- *DFSMS/MVS Storage Management Library: Implementing System-Managed Storage, SC26–3123*
- *DFSMS/MVS: Macro Instructions for Data Sets, SC26-4913*
- *DFSMS/MVS: Managing Catalogs, SC26-4914*
- *DFSMS/MVS: Program Management, SC26-4916*
- *DFSMS/MVS: Storage Administration Reference for DFSMSdfp, SC26-4920*
- *DFSMS/MVS: Using Advanced Services, SC26-4921*

- *DFSMS/MVS: Utilities, SC26-4926*
- *MVS/DFP: Using Data Sets, SC26-4749*

**DFSORT™**
- *DFSORT Application Programming: Guide, SC33-4035*

**Distributed Relational Database Architecture™**
- *Data Stream and OPA Reference, SC31-6806*
- *IBM SQL Reference, SC26-8416*
- *Open Group Technical Standard*

  The Open Group presently makes the following DRDA® books available through its Web site at: www.opengroup.org
  - *DRDA Version 2 Vol. 1: Distributed Relational Database Architecture (DRDA)*
  - *DRDA Version 2 Vol. 2: Formatted Data Object Content Architecture*
  - *DRDA Version 2 Vol. 3: Distributed Data Management Architecture*

**Domain Name System**
- *DNS and BIND, Third Edition, Paul Albitz and Cricket Liu, O'Reilly, ISBN 1-56592-512-2*

**Education**
- *IBM Dictionary of Computing, McGraw-Hill, ISBN 0-07031-489-6*
- *1999 IBM All-in-One Education and Training Catalog, GR23-8105*

**Enterprise System/9000® and Enterprise System/3090™**
- *Enterprise System/9000 and Enterprise System/3090 Processor Resource/System Manager Planning Guide, GA22-7123*

**High Level Assembler**
- *High Level Assembler for MVS and VM and VSE Language Reference, SC26-4940*
- *High Level Assembler for MVS and VM and VSE Programmer's Guide, SC26-4941*

**Parallel Sysplex Library**
- *OS/390 Parallel Sysplex Application Migration, GC28-1863*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28-1862*
- *OS/390 Parallel Sysplex Overview: An Introduction to Data Sharing and Parallelism, GC28-1860*
- *OS/390 Parallel Sysplex Systems Management, GC28-1861*
- *OS/390 Parallel Sysplex Test Report, GC28-1963*
- *System/390 9672/9674 System Overview, GA22-7148*

**ICSF/MVS**
- *ICSF/MVS General Information, GC23-0093*

**IMS**
- *IMS Batch Terminal Simulator General Information, GH20-5522*
- *IMS Administration Guide: System, SC26-9420*
- *IMS Administration Guide: Transaction Manager, SC26-9421*
- *IMS Application Programming: Database Manager, SC26-9422*
- *IMS Application Programming: Design Guide, SC26-9423*
- *IMS Application Programming: Transaction Manager, SC26-9425*
- *IMS Command Reference, SC26-9436*
- *IMS Customization Guide, SC26-9427*
- *IMS Install Volume 1: Installation and Verification, GC26-9429*
- *IMS Install Volume 2: System Definition and Tailoring, GC26-9430*
- *IMS Messages and Codes, GC27-1120*
- *IMS Utilities Reference: System, SC26-9441*

**ISPF**
- *ISPF V4 Dialog Developer's Guide and Reference, SC34-4486*
- *ISPF V4 Messages and Codes, SC34-4450*
- *ISPF V4 Planning and Customizing, SC34-4443*
- *ISPF V4 User's Guide, SC34-4484*

**Language Environment**
- *Debug Tool User's Guide and Reference, SC09-2137*

**National Language Support**
- *IBM National Language Support Reference Manual Volume 2, SE09-8002*

**NetView**
- *NetView Installation and Administration Guide, SC31-8043*
- *NetView User's Guide, SC31-8056*

**Microsoft® ODBC**
- *Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference, Microsoft Press, ISBN 1-57231-516-4*

**OS/390**
- *OS/390 C/C++ Programming Guide, SC09-2362*
- *OS/390 C/C++ Run-Time Library Reference, SC28-1663*

- *OS/390 C/C++ User's Guide, SC09-2361*
- *OS/390 eNetwork Communications Server: IP Configuration, SC31-8513*
- *OS/390 Hardware Configuration Definition Planning, GC28-1750*
- *OS/390 Information Roadmap, GC28-1727*
- *OS/390 Introduction and Release Guide, GC28-1725*
- *OS/390 JES2 Initialization and Tuning Guide, SC28-1791*
- *OS/390 JES3 Initialization and Tuning Guide, SC28-1802*
- *OS/390 Language Environment for OS/390 & VM Concepts Guide, GC28-1945*
- *OS/390 Language Environment for OS/390 & VM Customization, SC28-1941*
- *OS/390 Language Environment for OS/390 & VM Debugging Guide, SC28-1942*
- *OS/390 Language Environment for OS/390 & VM Programming Guide, SC28-1939*
- *OS/390 Language Environment for OS/390 & VM Programming Reference, SC28-1940*
- *OS/390 MVS Diagnosis: Procedures, LY28-1082*
- *OS/390 MVS Diagnosis: Reference, SY28-1084*
- *OS/390 MVS Diagnosis: Tools and Service Aids, LY28-1085*
- *OS/390 MVS Initialization and Tuning Guide, SC28-1751*
- *OS/390 MVS Initialization and Tuning Reference, SC28-1752*
- *OS/390 MVS Installation Exits, SC28-1753*
- *OS/390 MVS JCL Reference, GC28-1757*
- *OS/390 MVS JCL User's Guide, GC28-1758*
- *OS/390 MVS Planning: Global Resource Serialization, GC28-1759*
- *OS/390 MVS Planning: Operations, GC28-1760*
- *OS/390 MVS Planning: Workload Management, GC28-1761*
- *OS/390 MVS Programming: Assembler Services Guide, GC28-1762*
- *OS/390 MVS Programming: Assembler Services Reference, GC28-1910*
- *OS/390 MVS Programming: Authorized Assembler Services Guide, GC28-1763*
- *OS/390 MVS Programming: Authorized Assembler Services Reference, Volumes 1-4, GC28-1764, GC28-1765, GC28-1766, GC28-1767*
- *OS/390 MVS Programming: Callable Services for High-Level Languages, GC28-1768*
- *OS/390 MVS Programming: Extended Addressability Guide, GC28-1769*
- *OS/390 MVS Programming: Sysplex Services Guide, GC28-1771*
- *OS/390 MVS Programming: Sysplex Services Reference, GC28-1772*
- *OS/390 MVS Programming: Workload Management Services, GC28-1773*
- *OS/390 MVS Routing and Descriptor Codes, GC28-1778*
- *OS/390 MVS Setting Up a Sysplex, GC28-1779*
- *OS/390 MVS System Codes, GC28-1780*
- *OS/390 MVS System Commands, GC28-1781*
- *OS/390 MVS System Messages Volume 1, GC28-1784*
- *OS/390 MVS System Messages Volume 2, GC28-1785*
- *OS/390 MVS System Messages Volume 3, GC28-1786*
- *OS/390 MVS System Messages Volume 4, GC28-1787*
- *OS/390 MVS System Messages Volume 5, GC28-1788*
- *OS/390 MVS Using the Subsystem Interface, SC28-1789*
- *OS/390 Security Server External Security Interface (RACROUTE) Macro Reference, GC28-1922*
- *OS/390 Security Server (RACF) Auditor's Guide, SC28-1916*
- *OS/390 Security Server (RACF) Command Language Reference, SC28-1919*
- *OS/390 Security Server (RACF) General User's Guide, SC28-1917*
- *OS/390 Security Server (RACF) Introduction, GC28-1912*
- *OS/390 Security Server (RACF) Macros and Interfaces, SK2T-6700 (OS/390 Collection Kit ), SK27-2180 (OS/390 Security Server Information Package )*
- *OS/390 Security Server (RACF) Security Administrator's Guide, SC28-1915*
- *OS/390 Security Server (RACF) System Programmer's Guide, SC28-1913*
- *OS/390 SMP/E Reference, SC28-1806*
- *OS/390 SMP/E User's Guide, SC28-1740*
- *OS/390 Support for Unicode: Using Conversion Services, SC33-7050*
- *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 DCE Administration Guide, SC28-1584*
- *OS/390 DCE Introduction, GC28-1581*
- *OS/390 DCE Messages and Codes, SC28-1591*
- *OS/390 UNIX System Services Command Reference, SC28-1892*
- *OS/390 UNIX System Services Messages and Codes, SC28-1908*
- *OS/390 UNIX System Services Planning, SC28-1890*
- *OS/390 UNIX System Services User's Guide, SC28-1891*
- *OS/390 UNIX System Services Programming: Assembler Callable Services Reference, SC28-1899*

**IBM Enterprise PL/I for z/OS and OS/390**
- *IBM Enterprise PL/I for z/OS and OS/390 Language Reference, SC26-9476*
- *IBM Enterprise PL/I for z/OS and OS/390 Programming Guide, SC26-9473*

**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*

**RAMAC and Enterprise Storage Server**
- *IBM RAMAC Virtual Array, SG24-4951*
- *RAMAC Virtual Array: Implementing Peer-to-Peer Remote Copy, SG24-5338*
- *Enterprise Storage Server Introduction and Planning, GC26-7294*

**Remote Recovery Data Facility**
- *Remote Recovery Data Facility Program Description and Operations, LY37-3710*

**Storage Management**
- *DFSMS/MVS Storage Management Library: Implementing System-Managed Storage, SC26-3123*
- *MVS/ESA Storage Management Library: Leading a Storage Administration Group, SC26-3126*
- *MVS/ESA Storage Management Library: Managing Data, SC26-3124*
- *MVS/ESA Storage Management Library: Managing Storage Groups, SC26-3125*
- *MVS Storage Management Library: Storage Management Subsystem Migration Planning Guide, SC26-4659*

**System/370™ and System/390**
- *ESA/370 Principles of Operation, SA22-7200*
- *ESA/390 Principles of Operation, SA22-7201*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28-1210*

**System Network Architecture (SNA)**
- *SNA Formats, GA27-3136*
- *SNA LU 6.2 Peer Protocols Reference, SC31-6808*
- *SNA Transaction Programmer's Reference Manual for LU Type 6.2, GC30-3084*
- *SNA/Management Services Alert Implementation Guide, GC31-6809*

**TCP/IP**
- *IBM TCP/IP for MVS: Customization & Administration Guide, SC31-7134*
- *IBM TCP/IP for MVS: Diagnosis Guide, LY43-0105*
- *IBM TCP/IP for MVS: Messages and Codes, SC31-7132*
- *IBM TCP/IP for MVS: Planning and Migration Guide, SC31-7189*

**VS COBOL II**
- *VS COBOL II Application Programming Guide for MVS and CMS, SC26-4045*
- *VS COBOL II Application Programming: Language Reference, GC26-4047*
- *VS COBOL II Installation and Customization for MVS, SC26-4048*

**VS Fortran**
- *VS Fortran Version 2: Language and Library Reference, SC26-4221*
- *VS Fortran Version 2: Programming Guide for CMS and MVS, SC26-4222*

**VTAM**
- *Planning for NetView, NCP, and VTAM, SC31-8063*
- *VTAM for MVS/ESA Diagnosis, LY43-0069*
- *VTAM for MVS/ESA Messages and Codes, SC31-6546*
- *VTAM for MVS/ESA Network Implementation Guide, SC31-6548*
- *VTAM for MVS/ESA Operation, SC31-6549*
- *VTAM for MVS/ESA Programming, SC31-6550*
- *VTAM for MVS/ESA Programming for LU 6.2, SC31-6551*
- *VTAM for MVS/ESA Resource Definition Reference, SC31-6552*

# Index

# Readers’ Comments — We’d Like to Hear from You

**DB2 Universal Database for OS/390 and z/OS**
**Administration Guide**
**Version 7**

**Publication No. SC26-9931-01**

**Overall, how satisfied are you with the information in this book?**

| | Very Satisfied | Satisfied | Neutral | Dissatisfied | Very Dissatisfied |
|---|---|---|---|---|---|
| Overall satisfaction | □ | □ | □ | □ | □ |

**How satisfied are you that the information in this book is:**

| | Very Satisfied | Satisfied | Neutral | Dissatisfied | Very Dissatisfied |
|---|---|---|---|---|---|
| Accurate | □ | □ | □ | □ | □ |
| Complete | □ | □ | □ | □ | □ |
| Easy to find | □ | □ | □ | □ | □ |
| Easy to understand | □ | □ | □ | □ | □ |
| Well organized | □ | □ | □ | □ | □ |
| Applicable to your tasks | □ | □ | □ | □ | □ |

**Please tell us how we can improve this book:**

Thank you for your responses. May we contact you? □ Yes □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name

Address

Company or Organization

Phone No.

**Readers' Comments — We'd Like to Hear from You**
SC26-9931-01

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

# BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department HHX/H3
PO BOX 49023
SAN JOSE CA
U. S. A.
95161-9023

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Program Number:  5675-DB2

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.